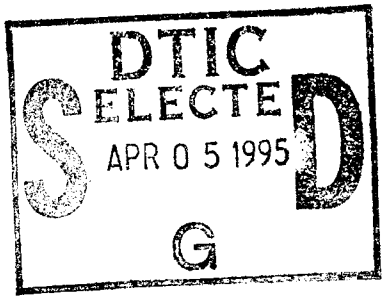
Task/Subtask IV02.1
CDRL Sequence A014–005
15 January 1995

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

Technical Papers: SWSC Domain Engineering Experience

Contract No. F19628–93–C–0129
Task IV02 – Megaprogramming Transition Support



Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731–2116

Prepared by:

Loral Federal Systems
700 North Frederick Avenue
Gaithersburg, MD 20879

Task/Subtask IV02.1
CDRL Sequence A014–005
15 January 1995

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

Technical Papers: SWSC Domain Engineering Experience

Contract No. F19628–93–C–0129
Task IV02 – Megaprogramming Transition Support

Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731–2116

Prepared by:

Loral Federal Systems
700 North Frederick Avenue
Gaithersburg, MD 20879

REPORT DOCUMENTION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1/15/95	Informal Technical Report

4. TITLE AND SUBTITLE

SWSC Domain Engineering Experience

5. FUNDING NUMBERS

F19628-93-C-0129

6. AUTHOR(S)

Brian Bulat, Loral Federal Systems - Gaithersburg

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Loral Federal Systems
700 North Frederick Avenue
Gaithersburg, MD 20879

8. PERFORMING ORGANIZATION REPORT NUMBER

A014-005

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Electronic Systems Center/ENS
Air Force Materiel Command, USAF
5 Eglin Street, Building 1704
Hanscom Air Force Base, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

N/A

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Cleared for Public Release, Distribution is Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Space and Warning Systems Center (SWSC) in Colorado Springs maintains and develops Command and Control Systems for NORAD, USSPACECOM, and AFSPC. Twenty-seven "stovepipe" systems imbedding twenty-four languages and ten millions lines of code make for a maintenance nightmare and for a fertile ground for Megaprogramming, or Product Line Software Reuse. At the SWSC, the SCAI (Space Command and Control Architectural Infrastructure) project has been instituting Architecture-Based, Product-Line Software Reuse for the past two years. The SCAI project is the STARS/Loral/Air Force Megaprogramming Demonstration Project out of ARPA.

This paper presents the lessons learned in developing and using a Domain Architecture, a Domain Architectrue Framework, and a Domain Engineering Process, while developing a SCAI Space Surveillance Application.

14. SUBJECT TERMS

domain engineering, architecture, reuse, object, class, framework, process, megaprogramming, product line

15. NUMBER OF PAGES

52

16. PRICE CODE

N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Preface

This document was developed by the Loral Federal Systems - Gaithersburg, located at 700 North Frederick Avenue, Gaithersburg, MD 20879. Questions or comments should be directed to Brian Bulat at 719-554-6577 (Internet: bulatb@lfs.loral.com).

Presenter: Brian Bulat

Title: SWSC Domain Engineering Experience

Track: Architecture (7)

Day: Tuesday

Keywords: domain engineering, architecture, reuse, object, class, framework, process, megaprogramming, product line.

SWSC DOMAIN ENGINEERING EXPERIENCE

This paper describes some of the lessons the Space and Warning Systems Center (SWSC) Domain Engineering team has learned while working on a Megaprogramming effort. In order to set the context for the lessons learned, the SWSC Environment, Domain Architecture, and Domain Architecture Framework are described. Architecture Framework extends the concept of architecture to include the relationship of architecture to other domain artifacts (system specifications, code, etc.) and the processes that support the construction of the architecture and artifacts. Finally, the lessons learned over the last two years on the Space Command and Control Architectural Infrastructure Project (SCAI) are presented. The lessons have been abstracted so that their scope is wider than just the SCAI project.

1.0 MEGAPROGRAMMING

The Software Technology for Adaptable and Reliable Systems (STARS) program under the Advanced Research Projects Agency (ARPA), has evolved the concept of Megaprogramming [Trimble94], or Process Driven, Product-Line Software Reuse supported by an integrated Software Engineering Environment. Megaprogramming identifies families of applications that share common traits and characteristics (a product line). Each common trait or abstraction may result in a common Ada component that can be used to construct multiple applications in the domain. Megaprogramming requires that software be produced according to defined processes. These processes are grouped into two lifecycles. A Domain Engineering Lifecycle, under the aegis of a Domain Engineering Organization, identifies common components; an Application Engineering Lifecycle uses the common components to construct specific systems. A Domain Management organization makes the cost decisions and scheduling decisions related to when common components are to be constructed and then integrated into individual applications.

2.0 SWSC ENVIRONMENT

The Space and Warning Systems Center (SWSC) in Colorado Springs maintains and modifies Command and Control Systems for NORAD, USSPACECOM, and AFSPC. Currently the SWSC is responsible for twenty-seven "stovepipe" systems, comprising over 10 million lines of code developed in twenty-four different languages on a variety of hardware platforms. This maintenance nightmare is a fertile environment for introducing Megaprogramming. At the SWSC, the SCAI (Space Command and Control Architectural Infrastructure) project has been instituting Megaprogramming for the past two years. The SCAI project is the STARS/Loral/Air Force Megaprogramming Demonstration Project out of ARPA.

The SCAI approach (see Figure 1) to the Product-Line Software Reuse aspect of megaprogramming, is to use a software process called Domain Analysis to create a Domain Specific Software Architecture, to which all individual systems in the domain/family are mapped. SCAI interprets the Domain and Application Lifecycles as being very tightly coupled: All Domain Analysis occurs in the Domain Engineering Lifecycle, but all component construction occurs in the Application Lifecycle. Because of this tight coupling, the Software Development Process is referred to as the Domain Engineering/Application Engineering Process (DE/AE).

SCAI intends to demonstrate that the SWSC version of Megaprogramming will increase software quality while decreasing the cost of developing and maintaining families of related SWSC Command and Control systems.

The SCAI project is creating a Space Tracking and Warning Application in the Space Domain using the SCAI Megaprogramming Technologies. Preliminary results in the first intermediate delivery of the SCAI System indicate that over 50% of the code in the system is reused or generated, just including the services layer of the architecture (explained in following paragraphs). It is presumed that as the scope of the domain is extended to encompass more than Space Systems, the percentage of generated and reused code will dramatically increase. Quality results have not yet been reported.

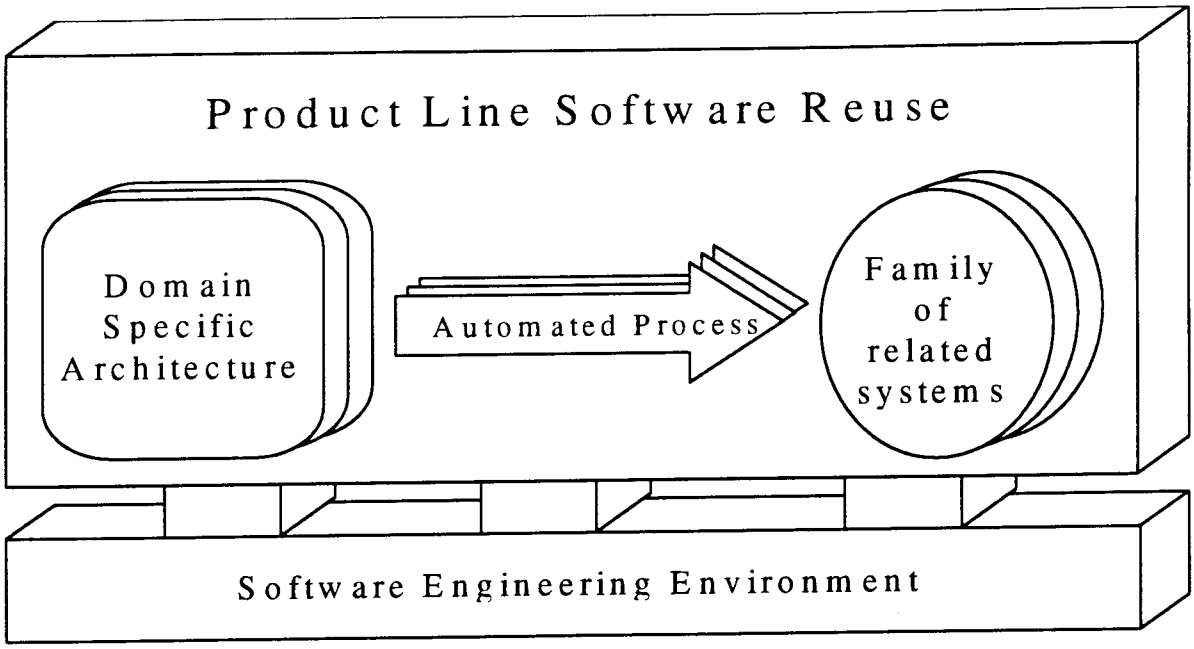


Figure 1. SCAI Megaprogramming

3.0 SCAI DOMAIN ARCHITECTURE

The SWSC attempted to determine a common architectural strategy for reengineering SWSC Command and Control Systems for several years prior to the SCAI project. This effort culminated in the RICC architectural infrastructure approach [Bristow93]. The so-called "Chip Diagram", Figure 2, represents the architectural concept at the start of the SCAI project. This concept had the SWSC C^2 system areas (Missile Warning, Space, Weather, Sensor Gateway, Air, and Intelligence) all being supported by a common infrastructure. The SCAI Project based the SCAI application on this infrastructure, which is enabled by the RICC tools.

The RICC, Reusable Integrated Command Center, is a set of run-time services supporting User Interfaces, Data Management, Message Handling, and Networks. Network services are a subset of system services in Figure 2. The RICC services are tailored to a particular system via a set of interactive Ada code generators, represented as Code Generator Tools in Figure 2.

This initial SCAI "chip" architecture was decomposed into a layered Domain Requirements Model (DRM) and a set of Application Architectural Models (AAM)s. The current scope of the DRM is the SWSC Space Domain; each AAM is specific to one system in the domain. The layered DRM is a modified Booch [Booch94] Class Model while the AAM is a network topology model and a mapping of application tasks to machines. These models, described below, comprise the ***SCAI Domain Architecture***.

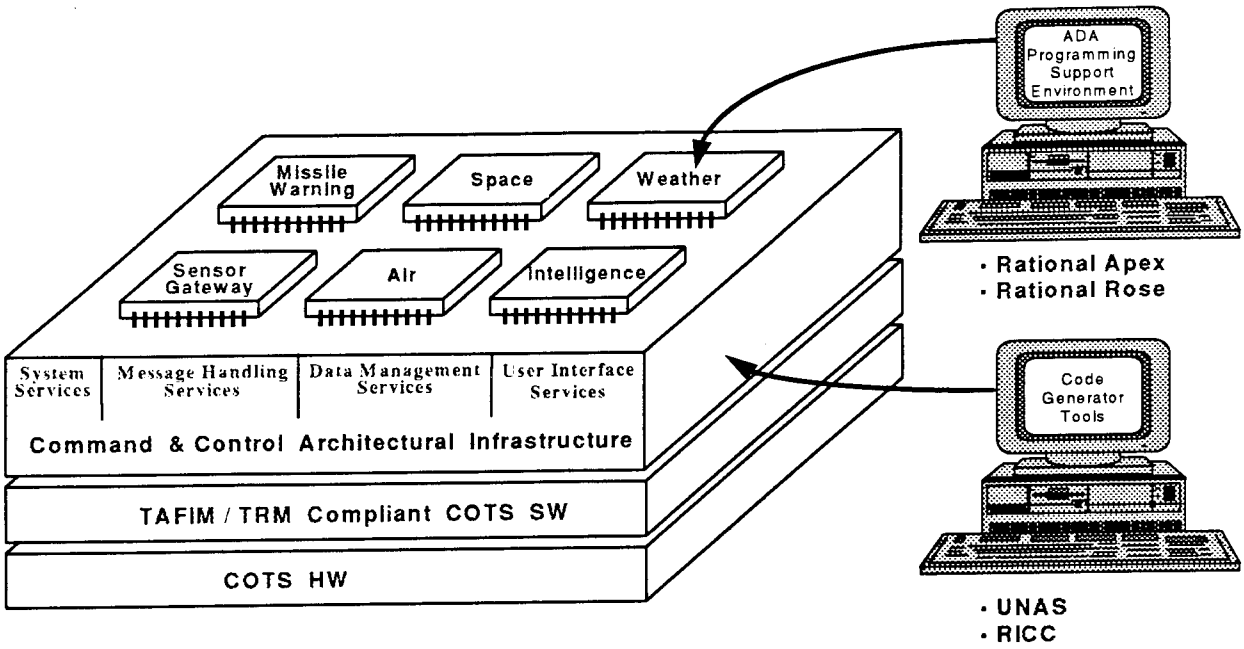


Figure 2. Initial C^2 Architectural Goal

3.1 DOMAIN REQUIREMENTS MODEL

The "chip model" was abstracted into a three layered object-oriented model called the Domain Requirements Model (DRM), Figure 3, by adding extra domain-specific model layers to identify types of reusable components other than the common service routines, and to further insulate the domain components from change.

3.1.1 Booch Model Composition

The DRM is a modified Booch [Booch94] Class Model. It contains a Class Diagram along with Class Templates and Object Scenario Diagrams. A Class Diagram identifies the static relationships between the classes in the domain, such as "a groundstation *tracks* a satellite", or a "maneuverable satellite is a subclass of satellite". Class Templates identify the details of class definitions: class operations (i.e., calculate a satellite orbit), class state data (i.e., satellite orbit definition), and class state transition diagrams (i.e., calculate a satellite orbit if new observations exist) if the class has complicated behavior. Object Scenario Diagrams depict single paths through the system, tracing the flow of control, as messages are passed from object to object, from one system boundary to the other. For instance:

(1) go to the groundstation object to get observations,

(2) pass the observations to the satellite object to calculate the new orbit,

(3) pass the new orbits to the orbit analyst object for viewing,

(4) put the newly calculated orbit in the satellite catalogue.

3.1.2 DRM Service Layer

The RICC Infrastructure Services, identified as the C^2 Service Layer in Figure 3, are services which are applicable to the whole C^2 domain. As new systems are analyzed and reengineered via the Domain Engineering / Application Engineering (DE/AE) process, the intention is to identify multiple instances of existing application services, and generalize them into a new C^2 Service Layer. These common Services act as servers to requesters (clients) in the layers above the C^2 Service Layer

COTS and GOTS products for use in the Service Layer are selected, as much as possible, so that they are compliant with the DoD TAFIM, Technical Architecture Framework for Information Management..

3.1.3 DRM Application Layer

Within the set of C^2 systems for which the SWSC is responsible, the Space Domain was chosen as the focus of the SCAI project. Common abstractions of the space domain are placed in the Application Layer. This Application Layer, shown as the middle layer of Figure 3, contains classes such as Groundstations, Satellites, and Observations, which are understandable to anybody familiar with the Space Domain. The classes imbed algorithms for things like orbit determination, and contain data such as orbits. Inheritance is used to identify system differences and commonalties in the domain. Commonalties are identified in "parent" classes while differences are identified in "child" classes.

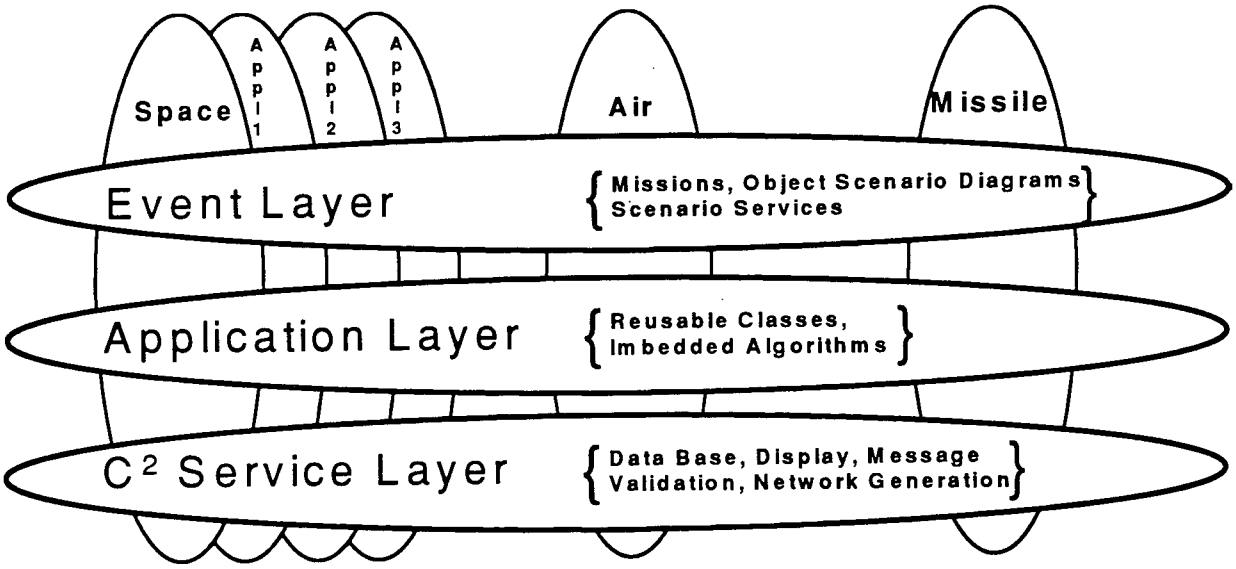


Figure 3. Domain Requirements Model

3.1.4 DRM Event Layer

Both the C^2 Service Layer and the Application Layer act as servers to the Event Layer. The Event Layer contains modified Booch Object Scenario Diagrams (OSD)s. Each class in the application layer may be included in many OSDs in the Event Layer. The OSDs act as a cross-check, to ensure that the classes defined in the application layer contain sufficient operations to handle all system functional requirements. A normal Booch OSD contains a single path through a system, tracing the flow of control, as messages are passed from object to object, from one system boundary to the other.

The SCAI DRM Event Layer is different than a set of Booch OSDs. Each Object Scenario Diagram (OSD), in the SCAI Event layer, traces a *set of paths* through the system, with messages connecting *all* the objects necessary to accomplish a single Space Mission. Space Mission examples are "Manually Determining an Orbit " or handling a "Satellite Reentry." These missions would typically be identified in a concept of operations document for a SWSC C^2 system, and would allow the Event Layer to be meaningful to Space Mission Experts. Each OSD, or mission, identifies every object needed by that mission. Each SCAI mission also relates to a meaningful, cohesive subset of the functional requirements for a system.

Each object within the OSD will eventually be transformed into an Ada Package or Generic Package, following SCAI coding rules. Therefore, each OSD also shows the necessary software needed to construct the Mission as an Ada Program, "on top" of the Service Layer run time services. Each SCAI OSD is a useful abstraction, understandable to a user of Space Systems, and shows how to construct an Ada Program to implement a SCAI Mission.

3.1.5 SCAI DRM versus other Domain Model Types

The DRM is a logical, machine-independent, representation of the common objects in all systems in the domain. So, DRM is really a misnomer. It is NOT a model of requirements, as one would never have included Services in a requirements model, because a requirements model is typically a model identifying a problem to solve, and not a design solution to that problem. Note also that the SCAI choice for a Domain Model is a high level design, including dynamic behavior as specified in the OSDs. Typical domain models [Diaz91-1] contain only static domain information. SCAI believes that a simple categorization scheme, embodied in a static model, does not provide enough information to determine if the reusable components (one component per class), can actually be used in the construction of the systems in the domain. The OSDs show how the classes (components) are used, and demonstrates that the components are sufficient to construct systems in the domain.

3.1.6 Reasons for DRM Layering

If the layering structure and rules for using the layers in the domain are consistent across that domain, then the DRM can be built iteratively. The characteristic

layering structure chosen for the DRM should be applicable to the whole SWSC domain.

The anticipated value of this layering is fourfold:

(1) The creation and isolation of a service layer allows the substitution of new, superior services in the future, while minimizing the impact to Space Application Layer Code. For instance, an object oriented data base could be substituted for the existing relational data base.

(2) The Service Layer makes it easier to be compliant with DoD TAFIM.

(3) New or modified missions can easily be created by reusing existing components in the Application Layer.

(4) The DRM layering scheme not only helps identify reuse opportunities, but also identifies the type of expertise that can be localized in product-line functional organizations, as explained in the following paragraphs.

Experts who understand the Missions in the SWSC domain need only understand the Mission Layer of the Domain Requirements Model, and need not be software engineers or system architects. These mission domain experts can collaborate to identify and abstract common missions across systems, and to project what new missions will be needed in the future.

Experts who understand orbits, trajectories, and the mathematics behind these topics, can be shared across appropriate SWSC supported missions, as part of a functional domain organization supporting the Application Layer.

Experts who understand typical software engineering tasks like relational data bases, or message parsing, can be considered as a pool available to all SWSC missions. They conceptually belong to the Service Layer of the SWSC, and should be responsible for upgrading and extending RICC services, or substituting services equivalent to those provided by RICC.

3.2 SCAI RELATIONSHIP BETWEEN DOMAIN AND APPLICATION MODELS

SCAI literature refers to both Domain and Application Requirements Models (DRMs and ARMs). Currently there is only a single DRM, which is being used to construct the SCAI application. Referring to Figure 4, classes (and resultant Ada Packages) are either specific to a single application or general to more than one application. Differences between applications are captured using child classes. Similarities between systems are captured using parent classes. OSDs in the Event

Layer are identified as either common to the whole domain or specific to a single (or several) system(s). For example, the Manual Orbit Determination OSD is common to Systems A, B, and C. It uses an Observation object which is also common to all three systems. However, Manual Orbit Determination also uses a Sensor object, which has different children for Systems A and B.

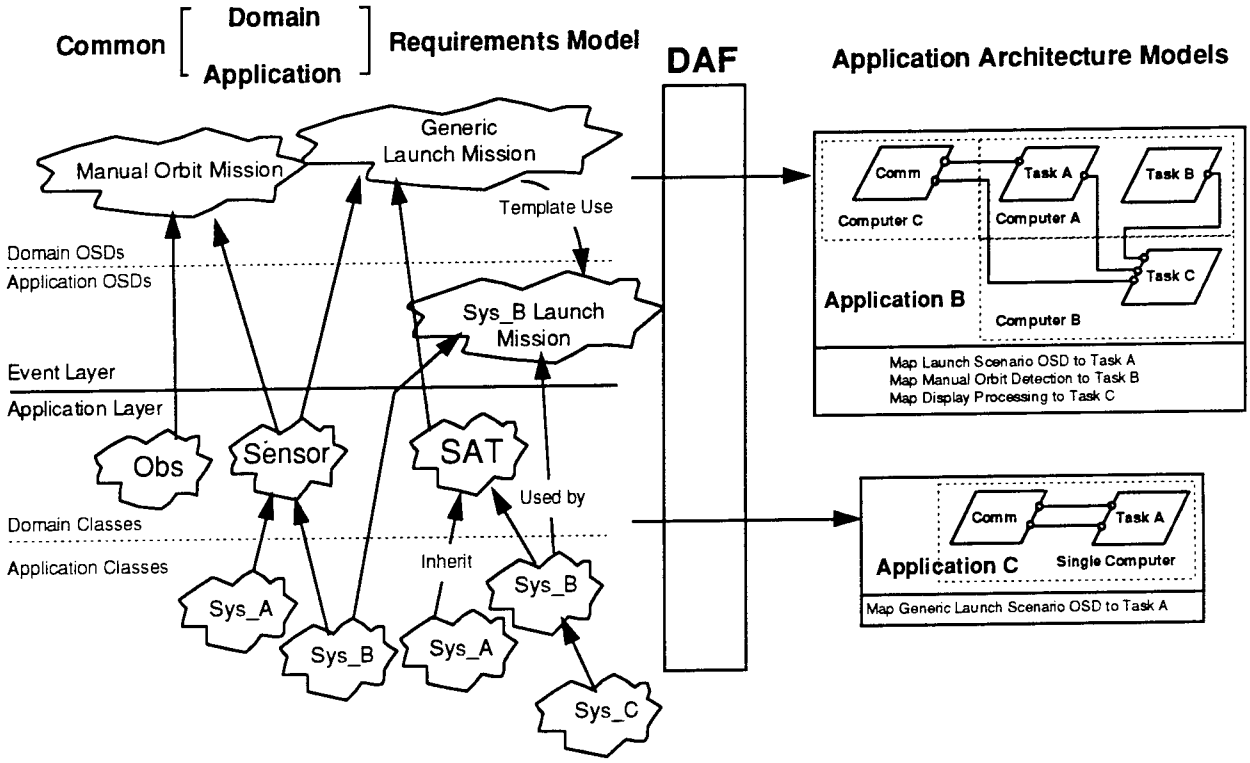


Figure 4. Domain/Application Model Relationship

The reason that application-specific classes are currently kept in the DRM is that when the next system is analyzed in the future, in order to extend the scope of the domain model, what is now an application specific class may be abstracted into a new class generic to several systems.

3.3 SCAI Application Architecture Model

Referring to Figure 4, the Domain Requirements Model (DRM) is mapped into a set of Application Architecture Models (AAM)s, one per system in the Space Domain. Each AAM is comprised of:

- a UNAS network topology model
- a UNAS to Mission Mapping Table

A Universal Network Architecture System (UNAS) model [TRW94], also called a Software Architecture Skeleton (SAS), is a network topology, comprised of tasks and

circuits. Circuits identify the paths across which messages flow between tasks, or units of asynchronous work. The UNAS model also maps UNAS tasks to machines.

UNAS is also a run time service that generates a TCP/IP network, independent of application code. The network model, and the run time services for the network, can be generated interactively off-line. Various configurations of the network can also be tested off-line, before application code has been generated, using "burn" statements estimating the load that the application will generate on the computer(s). UNAS allows architectural flexibility, because the same software components, consistent with the DRM, can be associated with many different network and hardware configurations and associated performance constraints. Also, software and physical architecture development can proceed in parallel.

The Application Architecture Model is complete when the Ada Programs, related to the Missions(OSDs), have been mapped to the UNAS Tasks.

DAF in Figure 4, stands for Domain Architecture Framework; the DAF defines the processes which transform the DRM into application specific physical architectures.

4.0 SCAI ARCHITECTURE FRAMEWORK

4.1 ITERATIVE SYSTEM DEVELOPMENT

A key lesson of the SCAI project is that both individual system models, and domain models/architectures, need to be built up iteratively. The connectivity between Ada Components in the DRM is the same as in the AAM; the AAM can be tested off-line by "injecting" simulated messages into the AAM. Therefore, the validity of the DRM can be tested using UNAS, and errors can be uncovered by executing the DRM before coding has proceeded, at the time in the life-cycle when errors cost the least to correct. Also, AAM development can proceed in parallel with Ada Package coding, compressing the time that it takes to develop a system.

4.2 ITERATIVE PROCESS DEVELOPMENT

Megaprogramming mandates the specification and the use of formal processes for the development and maintenance of SCAI models/architectures. Further SCAI experience has also shown the need to iteratively build up these processes and validate them with experience, exactly as with the models/architecture. Several times, as the architecture of the SCAI system has changed, the process has had to change, and vice-versa.

4.3 DEFINITION OF ARCHITECTURE FRAMEWORK

This intimate, synergistic nature between process and architecture must be extended to those other SCAI products: functional specifications, and Ada code, that

have relationships with the architecture, especially because of the iterative nature of the development process. Expressing the relationship between process, architecture, functional specifications, and code is done in the SCAI Architecture Framework. An architecture framework extends domain architecture by:

- including the definition of system specifications and code
- including the processes which build the architecture and system specifications and code

Though not discussed in this document, metrics are also included in the SCAI Architecture Framework to evaluate the quality of the artifacts and architecture.

A SCAI goal was to base the architecture framework on processes with a successful track record. The three major processes that were incorporated into the SCAI Architecture Framework were:

- Booch Object Oriented Analysis for DRM building [Booch93, Booch94]
- Cleanroom Software Development for Requirements Specification, Class translation into Ada Code, and Statistical Testing. [Pal92A]
- TRW Ada Process Model for building the Application Architecture Models and for translating the Service Layer generic services into an application specific product. [Pal92B]

The mapping between the major domain artifacts, and the processes used to build those artifacts, is shown in Figure 5. The mapping is shown to the level of each layer in the architecture. Note that Figure 5 also identifies the fact that SCAI systems have three different architectural views, one per column:

- A Functional View
- A Logical View (both dynamic and static logical relationships)
- A Physical View

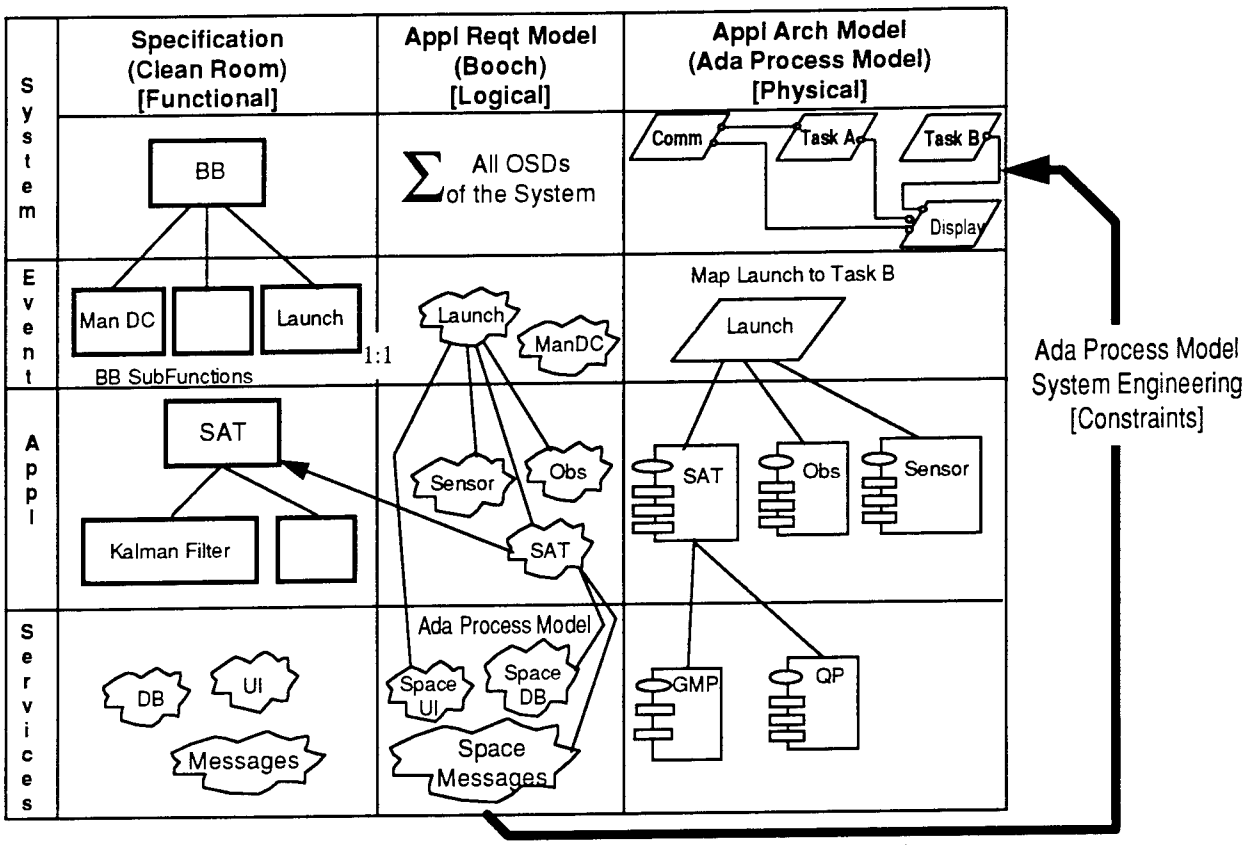


Figure 5. Architecture Framework

System-level artifacts are identified by the top row in Figure 5:

- A system-level functional specification
- A system-level logical model
- A system-level physical mode

Each system level artifact is decomposed into the three layers (Event, Application, and Services) that have been defined previously.

Implicitly, the domain-level logical model (DRM) can also be considered to occupy the system-level logical model spot in the framework; this DRM extends the scope of an individual system to the entire Space Domain, and contains the set of OSD's for the entire Domain.

4.1 Process for Building the Logical Model

System and domain logical models are created, as previously stated, using Booch Domain Analysis. The inputs to this analysis are: Business Process Models,

System Specification Documents, System Design Documents, System ICDs, System Code, and System Operational Concepts Documents.

Roughly speaking, analysis involves proposing strawman generic models, examining existing systems documentation and encoding this information as classes within the models, identifying similarities and differences between the systems, and refining the generic model to accommodate existing systems. Scenarios are used to prove that the generic class models are correct.

4.1.1 Restrictions on SWSC Domain Analysis

At the SWSC, Space Systems that are restructured to conform to the Architecture Framework must still satisfy their original requirements. In general, the same inbound and outbound messages must be processed, and the same display screens must interact with the system users via the same dialogues. Invoked Space Algorithms must have the same accuracy. Therefore, once a good set of Application Objects has been abstracted (grouping and abstracting needed Space Algorithms), the validity of the abstractions is tested by creating object scenario diagrams in the event layer that pass inbound and outbound messages exactly according to current systems functional specifications.

4.1.2 Application Classes into Service Classes

As the Booch Analysis proceeds, common Application Classes are tested to see if they might conform to rules that identify service classes. For instance, if two applications (Space, Missile Warning) both contain alarm processing, then that processing may be isolated in the Services Layer, at the discretion of Domain Management.

4.1.3 Pattern Recognition

The Event layer invokes each Application "Server" that is needed to execute a Mission. Common Services associated with all Missions in the Event Layer (such as error processing) are encapsulated in new class utilities. Also, certain dynamic patterns of use for every Mission have emerged. For instance, whenever a UNAS task is initialized or terminated because a command associated with a Mission has started or terminated, the same type of processing routine is invoked across every mission. To uncover these common patterns of usage, a "pattern recognition" process is used in this layer, which is not object oriented, as well as standard Booch Analysis.

4.1.3 Extending the Domain Incrementally

Every time a new system is incrementally added to the scope of the domain analysis, each class and each mission are examined to see if they can be abstracted with the new domain analysis input data. A variety of options for integrating the new information into the DRM might occur:

- A new class might be added which is independent of existing classes. This does not affect existing Missions.
- An existing class is identified as generalizable, but will not be reabstracted because of the cost of modifying existing systems. A temporary" child class for the new system, tagged as belonging to the new system, will be added to the existing class. The old class will also be tagged as awaiting abstraction.
- An existing class will be modified. New Ada Code will be written for the new class. Any Missions using the class will be retested.

4.1.4 Domain Management

Domain Engineering makes the technical decisions related to extending and abstracting the DRM. An organization called Domain Management makes the scheduling and cost decisions as to whether software should be generalized to reflect these changes and whether the software should be updated and then retested in any or all applications to which it is related.

4.2 Process for System Specification

Cleanroom Software Engineering is used to create the Functional View of SCAI Systems. Cleanroom is a formal method which provides a tightly integrated approach from specification preparation through software development to software certification , bringing engineering rigor and intellectual control to the process [Mills86]. Intellectual control is the ability to clearly understand and describe the problem at hand at the desired level of abstraction. Cleanroom introduces the concept of statistical testing to ensure the developed system meets the users requirements to any level of statistical accuracy. System specifications are coded using Black boxes which explain how stimulus histories are mapped into responses. Black Boxes are then decomposed into State Boxes, which replace stimulus histories with internal state data. The Clear Boxes are then decomposed into Clear Boxes which elaborate the functions that translate stimuli into responses. The process iterates as Clear Boxes are decomposed into new Black Box Subfunctions. This process is called Box Structured Decomposition.

SCAI System Specifications are built using the Cleanroom Software Engineering Process. A system black box (BB) artifact is created, using both the DRM and the same inputs (other than System Design and Code) as used by Domain Engineering. A system BB contains all system stimulus histories (input messages with conditions under

which the message will be accepted), and corresponding system responses, as well as functions which map these stimulus histories into the system responses.

4.3 Process for Functional Decomposition

System BBs are decomposed, via Box Structured Decomposition, into a set of BB subfunctions. Each BB subfunction must map directly into a mission in the DRM Event Layer. Each input message processed by a BB subfunction must be identified in the corresponding mission OSD.

Associated with each stimulus in each BB subfunction, is a probability of the stimulus occurring. The probability is used to evolve a test plan for the subfunction/mission/Ada program that will guarantee that the mission is accurate to any degree of desired accuracy.

Missions are not decomposed using BB decomposition beyond the subfunction level. The decomposition of a subfunction is accomplished using domain analysis, and recorded within the logical model by identifying the classes that occur in the OSD for the mission. BB decomposition cannot be used to identify these classes, because they are generic to the domain, and may imbed methods that do not relate specifically to any one subfunction; in other words, they are reusable across the Space Domain.

Once the Application Classes have been identified (Application Layer, Logical Model), they are translated into Ada Packages using BB decomposition, in order to guarantee the accuracy of the very complicated Space Algorithms.

4.4 Process for Building a Physical Model

As explained previously, the Architecture Model contains UNAS Tasks imbedded with Ada Programs representing Missions in the Event Layer. Application Layer Classes identified in OSDs are included (in the form of Ada Packages) within these Mission Ada Programs. The system level physical model is represented by the upper right box in Figure 5.

The Architecture Model is built using methodology defined in the Ada Process Model, a systems engineering process specific to Command and Control Systems. The TRW Ada Process Model [Royce89,90a,b;PAL92] is an iterative, demonstration-based, code generator-based, development technique particular to Command and Control Systems

Once basic system missions have been defined in the logical model, the BB subfunctions have been defined in the functional specification, and performance constraints have been identified for the system, Ada Process Model (APM) iteratively builds proposed physical architectures, identifying flaws in each proposed architecture, and then correcting those errors.

APM addresses typical system engineering concerns by performing "trade studies" that balance cost against hardware and storage resources.

The generic Common Services identified in the Service Layer of the Functional column of Figure 5, must be made system-specific. For instance, each display format for the SCAI Application contains a command line, which must be parsed. Therefore, a parsing service is added to display services for the SCAI Application. All system-specific services for a single application reside in the Service Layer under the Logical Column. An encapsulating class is defined for each particularized service. The encapsulated classes are used to reference these classes in the Logical Model Event and Application Layers.

4.5 Related Process Paper

Note that the process for integrating the Logical, Physical, and Functional development processes on the SCAI Project will be discussed in the paper "Product Line Software Development" by David J. Bristow also under the Architecture Track at STC.

5.0 Lessons

The following lessons in Domain Engineering relate to both the creation of the Architecture Framework, and its integrated processes, and to the execution of those processes on the SCAI project. The lessons have been abstracted so that they have broader scope than just to the SCAI project, or just to the Space Domain. These lessons, among other SCAI Project lessons, will be formally delivered in "AF/STARS Demonstration Project Experience Report Version 2.0," Contract No. F19628-93-C-0129, CDRL Sequence A011-002D.

5.1 Lesson: The DE Process Must Allow for Iterative Domain Knowledge Acquisition.

Product Line Software Development implies two software life cycles: one life cycle developing generalized domain products and a parallel life cycle developing individual applications which are constructed from the domain life cycle products. Obviously, domain products must exist prior to their use on the application. Problems associated with the prior construction of all domain components are:

- Great upfront DE costs not associated with developing any product. (As DoD shrinks, these kind of costs are increasingly difficult to justify).
- Generalized Models and generalized components can only be validated through their use on real systems. Even within a single system, a reusable class must be validated in each of the contexts in which it is

used. Monolithic "water-fall" systems development has largely been discredited vis-à-vis more iterative approaches to modeling and systems development.

- A large domain, such as Command and Control, may contain very many systems. In spite of the fact that all these systems have much in common, it is unlikely that DE can be initially accomplished with the scope of every one of these systems, before the need to deliver the first rearchitected system occurs.

For the above reasons, the DE process must allow for the iterative acquisition of domain knowledge. The SCAI Architecture Framework was constructed from processes which are inherently iterative. Therefore, the overall process is iterative. As new systems are analyzed, and the scope of the DRM is extended, more domain missions are identified, new classes are created, and existing classes are generalized. Newly generalized classes are reinserted into existing missions, and the missions are retested.

The cost and time to perform the Domain Analysis (DA) of the Space Domain, on the SCAI Project, was underestimated and the DA analysis is not currently complete. However, the ability to continue to build generic software has not been halted, because the DA process is iterative.

5.2 Lesson: Interpret the STARS "Two-Lifecycle Model" as a Single Integrated DE/AE Process.

The SCAI team has found it necessary to interpret the STARS "Two-Lifecycle Model" to show the very close interaction between Domain Engineering and Application Engineering, in order to constantly validate the domain models against real applications in the domain. In the original STARS model, Figure 6, the parallel Domain Engineering Lifecycle creates a domain design and domain implementation. These generalized artifacts must be kept consistent with application specific systems designs and systems implementations in the Application Life Cycle.

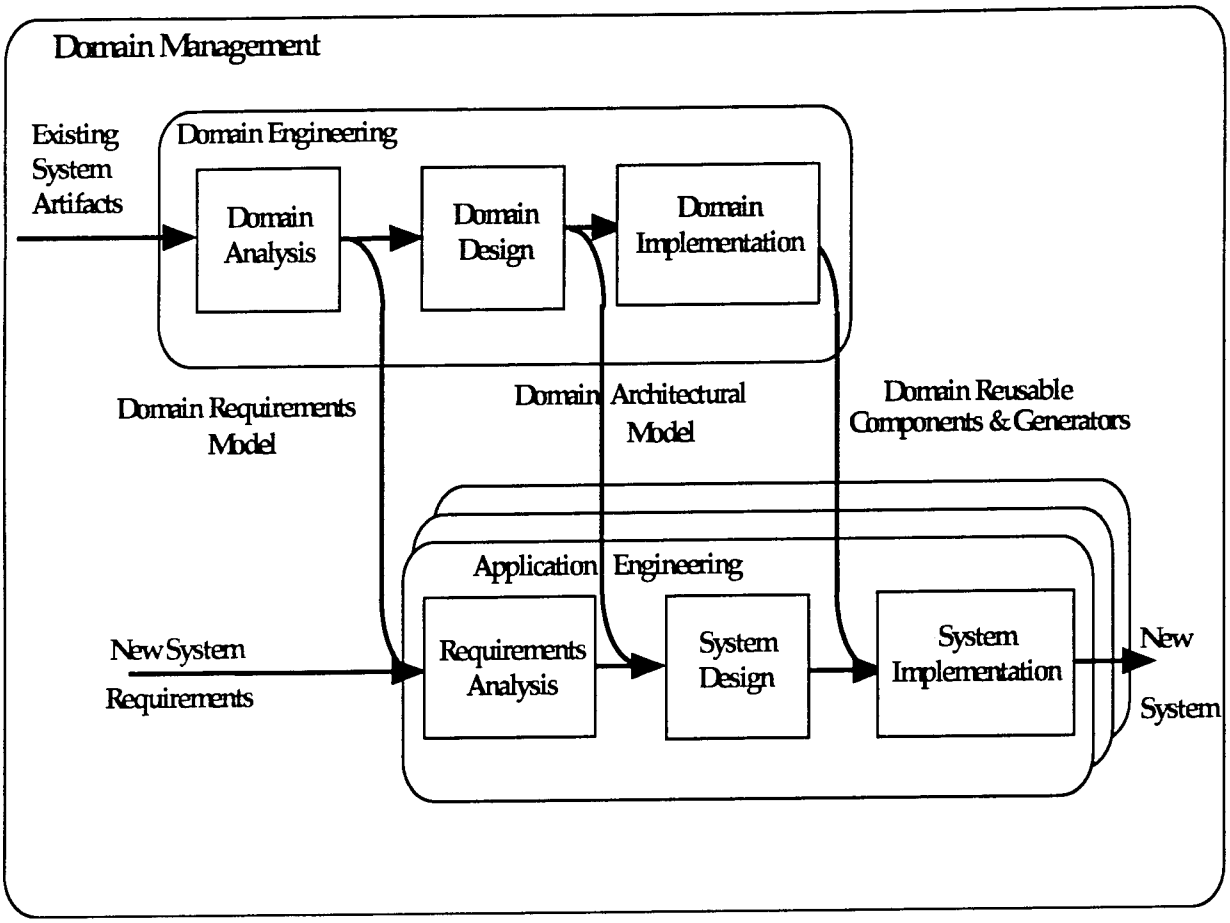


Figure 6. STARS Two Life-Cycle Model

To avoid serious configuration management problems, SCAI decided to maintain a single logical model, the DRM, used by both the Domain and Application Lifecycles, Figure 7. Applications would add detail to the model; The Domain Organization would add new classes when extending the scope of the domain, and potentially generalize existing classes within the previous scope of the domain. All system specifications and system components would point to this single DRM. Component generation, both generic to more than one system, and specific to a single system, would be assigned to the development Lifecycle of an existing Application. Testing for the generalized components would then have to occur in each of the applications to which the component applied. Test cases and infrastructure are maintained with each application, so retesting within an existing application should be routine. Again, the cost decision to actually reintegrate the generalized component within an existing application is a decision of the Domain Management organization

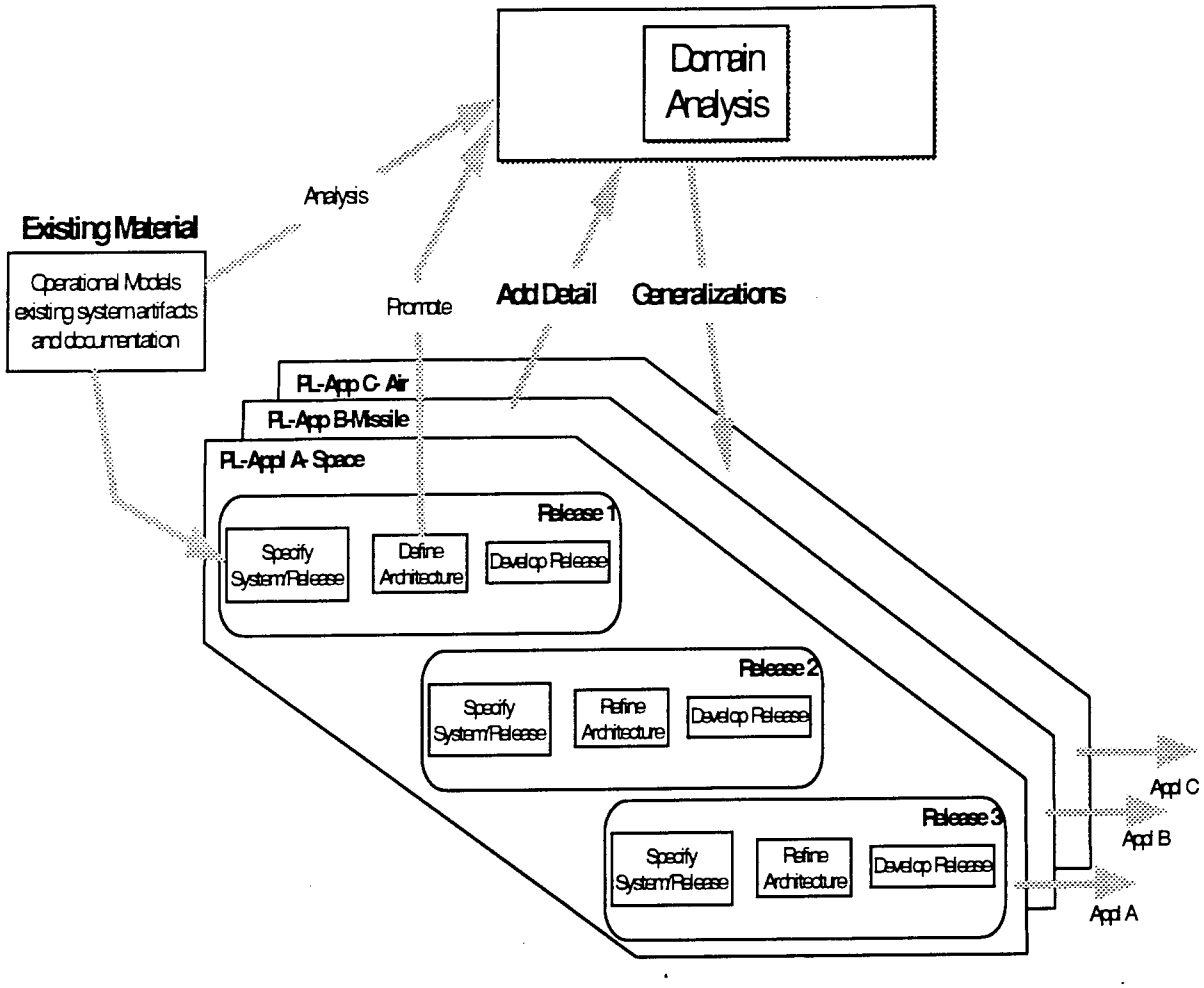


Figure 7. Modified Two Life-Cycle Model

5.3 LESSON: THE DOMAIN ANALYSIS PROCESS SHOULD BE APPLICABLE TO INDIVIDUAL APPLICATIONS.

Domain Analysis (DA) typically looks at multiple systems with the intent of discovering the similarities and the differences between those systems. The overlap of DA with Systems Analysis should be in identifying common abstractions across any scope of analysis; common abstractions will result in common code components. If the DA process is well-understood it should be applicable to a single system. When applied to a non-perfect, single system, the result of DA should be a simpler, well-structured, single system. Thus, even if an organization is initially unwilling to invest in DA and generalized component development across the whole extent of the domain, they may be willing to apply domain analysis to a single system, and still be able to judge the economic and technical value of the DA process. That the DA process is not affected by the scope of its application, is essential to doing incremental domain analysis as specified in Paragraph 4.1. This does not change the central purpose of domain analysis from identifying commonalties across multiple systems.

5.4 LESSON: USE VALIDATED TECHNOLOGIES FOR DOMAIN ENGINEERING

SCAI believes that Domain Specific Software Reuse may require dramatic cultural change, but not sophisticated new technologies. It has been shown previously that there is much overlap between Systems Engineering and Domain Engineering. Domain Specific Reuse should be implemented using technologies that are intuitively easy to understand, and have widespread use. Then, the cultural shock associated with megaprogramming will not be reinforced by a technological learning curve.

For example, existing Object Oriented methods (for instance, Booch Analysis) are sufficient for recording the results of DA. Superclasses, class categories, and polymorphism intrinsically identify high level abstractions/system commonalties. Subclasses identify system differences. Classes provide "real world" abstractions that do not require programming expertise to understand. At a high level of abstraction, a Satellite Class is easier to understand than a set of functionally decomposed orbital calculation algorithms. Satellite relationships to groundstations and orbits encoded in a class diagram provide much more information that a simple hierarchical decomposition.

Originally, SCAI Domain Analysis was going to be performed using the methodology of Ruben Prieto-Diaz [Diaz91-2]. Even though the methodology was very promising, SCAI was unwilling to use it because it did not have the track record of the more prosaic Object Oriented Methodology. However, key aspects of the methodology were incorporated into Booch, such as the bi-directional mapping of low level system abstractions to a high level architecture, and revisions.

SCAI Configuration management of domain and application artifacts has been accomplished using standard tools such as Rational CMVC, and does not require the use of a Software Reuse Library Mechanism. The Software Reuse Community originally thought that the key to software reuse was making components available in a Reuse Library Mechanism having sophisticated search mechanisms. However, SCAI recognizes that the DRM is the key to understanding and finding all domain software components.

5.5 LESSON: DOMAIN ANALYSIS SHOULD EXAMINE THE RELATIONSHIP OF PEOPLE/ORGANIZATIONS TO SYSTEMS

A restriction to SCAI Domain Analysis is that SCAI rearchitected systems must retain their existing external interfaces. SCAI DA stops at system boundaries. This restriction seriously degrades the ability of the Domain Engineering Organization to improve existing systems based on new technology. For example, in the Space Domain, several systems involve the Orbit Analyst in a complicated dialogue across multiple display screens in order to determine the orbit of a satellite that requires non-routine calculations. The original reason for creating the dialogue, was that these non-routine calculations require a great deal of compute power, and the human (Orbit

Analyst) was part of the loop to determine the situations in when the superior accuracy of the nonroutine calculations was critical. As hardware becomes faster, the nonroutine calculations can be performed every time, and the dialogue with the human can disappear altogether.

Human roles and computer roles in the overall "business process" for an organization need to be included in an accurate Domain Analysis. The SWSC currently does Business Process Modeling, but this activity is not integrated with SCAI Domain Engineering.

5.6 Lesson: A Domain Model Must Contain Behavioral Abstractions

SCAI requires that application functional specifications be kept consistent with the DRM. If this consistency is maintained, then SCAI systems will satisfy their functional requirements. Functional specifications show how system input messages are mapped to system output messages. It must be shown that when the same messages enter the DRM that a path exists through the DRM such that the same output messages will be generated as in the functional specification. This need to trace messages through the DRM requires that behavioral abstractions be included in the DRM.

OSDs define complete paths between objects in the DRM, when associated with State Transition Diagrams and Ada PDL to specify the logical conditions under which messages may flow. State Transition Diagrams and Ada PDL may also have to be associated with individual objects that have complicated behavior, to identify the logical conditions under which messages will be passed out of the object.

When all paths through the DRM are maintained as part of the DRM, then it can be considered "executable" and consistent with its requirements.

5.7 Lesson: Domain Analysis Should Produce the Simplest Reusable System.

Behavioral abstraction is also needed to evaluate the complexity of the DRM. Static Domain Analysis of the functionality that a domain must contain is not enough to ensure that reusable components can be combined into well-architected, simple to understand systems. DA should identify components that will result in the *simplest possible* reusable system within the context of the domain.

SCAI DA compares the complexity of the behavior encapsulated within classes (class state transition diagrams) relative to the complexity of the interfaces defined between those classes. Interfaces between classes are represented by OSDs as well as the State Transition Diagrams associated with each OSD. If the OSD is very complicated, requiring interfaces to many objects, then the mission represented by the OSD is tightly coupled. In order to loosely couple the OSD, the class definitions will be

reformulated. Standard Object Oriented Metrics are used to evaluate the complexity of the model.

5.8 Lesson: Processes Are Replaceable in an Architecture Framework

At least three different domain analysis processes have been used to abstract a set of common services for Command and Control Systems. Ruben Prieto-Diaz applied his Domain Analysis Process Model at Contel [Diaz91-2]. TRW Corporation applied an internal DA method to identify the RICC Services used by SCAI. Loral Corporation also used an internal method to abstract common services for its CCS-2000 Domain Specific Architecture. In all three cases, the same set of services were identified.

Booch Analysis was chosen to elaborate the logical architecture. However, other Object Oriented methods were examined during the SCAI preparation phase (Rumbaugh, Shlaer-Mellor, and others), and, from the method selection analysis, other methods also would have been good choices for decomposing the logical architecture.

Cleanroom Software Engineering was chosen to decompose the functional view of SCAI systems, but a variety of structured analysis techniques could have been substituted for Cleanroom.

The point is, that once an architecture framework has been defined for a domain, a variety of processes can be used to develop the artifacts within different layers of the Architecture Framework. The reason for choosing a particular process may simply be that personnel are familiar with that process.

5.9 Lesson: Layer the Domain Architecture to Insulate the Domain from Technological Change

When technological changes mandate system changes, systems should not have to be redeveloped from scratch. Big DoD systems are currently migrating from large mainframe computers to networked workstations. The cost of rearchitecting these Command and Control systems is proving exorbitant. The next such technology transition might be toward massively parallel computers.

Typically, a DRM would not contain a service layer, because activities such as querying a relational data base, are not normally part of an abstract, problem-state model. However, SCAI recognized that identifying common services, and placing them in a layer of the DRM not only simplified the domain analysis process, but facilitated the process of "encapsulating" these services. Once classes have been defined to encapsulate these services, the cost of replacing these services with equivalent services can be readily estimated, because all references to the services are graphically clear in the Class Diagram of the DRM.

An obvious future service replacement to the SCAI architecture, would be to replace the current relational data base with an object oriented data base, or to replace UNAS with a CORBA compliant networking service.

Future SCAI Domain Engineering plans call for abstracting the current DRM Event Layer into two layers: the current Event Layer plus a User Interface Layer. The User Interface Layer would totally isolate dialogues between a system and its users, to allow simple replacement of the Display service, and to identify common dialogues and simplify the way users interact with SCAI systems.

5.10 Lesson: Product-Line Organizations Should be Constructed According to the Architecture Framework

Contractors and Air Force personnel on the SCAI project have been co-located by subcontractor, and not by the process in the Architecture Framework that they are performing. Communication and synergy are lost because it is sometimes difficult for personnel executing the same process to "feed" off each others expertise.

Each Architecture Framework process requires a different set of skills and knowledge to perform. For example, executing the Ada Process Model requires skills in network architecture and systems engineering, while building a mission in the Event Layer of the DRM requires an understanding of space operations.

Product-line personnel should be organized into functional organizations based on the architecture and be co-located to facilitate information exchange, which would also avoid duplication of the Software Engineering Environment (SEE) facilities.

Organizational structures that parallel the architecture framework, enhance the communications needed to maximize reuse opportunities and deliver on the promised Megaprogramming productivity increases. This organizational approach also avoids duplication of effort.

The "other side of this coin" is that an architecture should be originally developed for *organizational* reasons as well as technical reasons. The SCAI Mission layer of the DRM was created partially to ensure that space operators could understand the domain without being forced to understand the mathematics of the Application layer and the Software Engineering of the Services Layer.

5.11 Lesson: Validation Data Is Associated with Missions, Not Reusable Components

Cleanroom Software Engineering uses probabilities associated with the likelihood of system stimuli occurring, and the "amount" of testing performed, to determine the statistical likelihood that programs will fail. SCAI Stimuli are associated with Missions. Therefore, validation data (test results, test cases, test data, etc.) relate to Missions; no statistical accuracy is associated with any specific reusable component.

Reusable components are *not* unit tested. Therefore, no contention is made, or can be made, about the validity of a reusable component outside the context of its use.

This is just another specific example of Domain Specific Reuse, where no contention is made that a domain component is reusable outside of the domain context identified by the domain model.

5.12 Lesson: Both Functional and Object Oriented Mind Sets are Required to Build a Space Architecture Framework

Space systems requirements are functionally stated; the technology to develop software and test it to a specified degree of statistical accuracy is functional. Most COTS and GOTS reusable software is functional. Therefore, for reuse in the present day, it is necessary to provide a functional view of the Architecture.

However, Object Oriented Modeling, at least according the SCAI philosophy, is the best way of decomposing a domain so that it can be *understood,* and is *reusable.* However, one lack in Object Oriented Modeling, according to some technologists, is that it is an informal modeling technique. Domains are not decomposed into precise, verifiable units. Therefore, there is a concerted effort to combine formal methods with Object Oriented Modeling, in order to add this precision. Cleanroom Software Engineering imbeds semi-formal methods for software generation, and therefore, remedies Object Oriented Modeling flaws.

5.13 Lesson: Megaprogramming Projects Still Need to Focus on Basic Engineering and Project Management Disciplines.

New technology is always viewed as a "silver bullet." However, no technology obviates the need for bright system engineering and management talent, and people who understand the domain of the product line. In fact, as the technologies become more sophisticated, the need to have talented people actually increases.

Although Megaprogramming brings to the table multiple concepts to improve the production of software systems, it does *not* replace some of the basic engineering and project management disciplines.

Megaprogramming extends the scope of software development, requiring that more and more people work together (though eliminating redundant personnel). Establishing an efficient organization with cooperating domain and multiple application organizations, all building software incrementally and cooperatively, is a formidable management challenge.

Regardless of how well defined domain components are, inserting those components into an efficient network architecture still requires sophisticated systems engineering talent.

5.14 Lesson: Domain Analysis Needs to be Supported by a Product Line Organization.

The SCAI project conducted domain analysis on the Space Domain which included the SPADOC 4C data base. SPADOC 4C is a Space System, partially deployed in Cheyenne Mountain, that was analyzed via SCAI domain analysis. A number of redundancies were uncovered in this data base, which presumably was constructed incrementally over the life of the SPADOC system. Due to the stovepipe organizational structure and resulting differences in priorities, it has been difficult to cooperate in this area (which could benefit both SCAI and SPADOC). We believe that organizational structuring based on the product-line approach would enhance the SWCS's ability to take advantage of Domain Analysis, increase the focus on commonalties between systems, avoid reporting boundaries and facilitate information sharing between current systems and new ones.

From the above small experience, deploying SCAI Megaprogramming technologies, will be far harder than inventing the SCAI Megaprogramming technologies in the first place.

5.15 Lesson: Pattern Recognition Worked for the SCAI Project.

It was presumed that reuse would occur in the Application Layer of the DRM as multiple missions in the Event Layer used the same Application Layer Classes. In fact, as missions were developed, not only was it found that as each mission was elaborated, the same common services were required to support the mission, but also that the services were applied in the same order. Thus, a whole set of reusable services were created, and were encapsulated in Class Categories called Event Control and Event Common. This type of discovery of reuse opportunities (through order of class invocation) is an example of a new theory of architectural reuse referred to in the literature as "Pattern Recognition."

5.16 Lesson: Critical Paths of the Physical Architecture Must be Iteratively Tested.

The Ada Process Model, incorporated into the AE/DE process, demands that a system, including its architecture, be iteratively built and tested. For example, the original SCAI Domain Architecture did not distribute the human-machine interface module to each workstation, creating too much message traffic on the network. This overload condition was identified because UNAS allowed for architectural testing before any significant application code had been written. This enabled the architect to correct the architecture without impact to the schedule. This is a specific example of the general need to iteratively develop and test the physical architecture.

5.17 Lesson: Ada Code Generators created significant amounts of reusable code.

Ada Code Generation, such as that provided by the Reusable Integrated Command Center (RICC) Tool set, has greatly decreased SCAI code development time. RICC infrastructure provided reuse across the entire space domain including user interface generation, message parsing/assembly, network control, and data base support. RICC was used to help elaborate user interfaces rapidly and gain customer acceptance. RICC provided 52% of the SCAI Release 1 code (19% reused, 33% generated).

5.19 Lesson: Megaprogrammers Must Understand Process Definition

Megaprogrammers not only need skills in software development, but also in process definition and improvement. Process definition must be recorded by those who will use it, so that they will enthusiastically accept the process, identify inefficiencies in it, and identify needed process improvements. On the SCAI Project, when software development organizations did not record the processes they used, they felt free to ignore those processes.

6.0 Summary

The SCAI Project focuses on how standard, proven software engineering technologies can be used for Domain Engineering, not on the need for unique new technologies.

SCAI emphasizes that Software Process and Software Architecture are intimately related, using the Architecture Framework to show the relationships.

SCAI recasts Domain Engineering as an *incremental* process that is *tightly coupled* with Application Engineering. Incremental applies to both the elaboration of the architecture and the elaboration of the process that builds the architecture.

Finally, SCAI emphasizes that Product Line Organizations are required to implement Megaprogramming and that these organizations should conform to the architecture used to subdivide the Domain.

REFERENCES

[Booch94] Booch G., ***Object-Oriented Analysis and Design***, 2nd Edition, Benjamin Cummings, 1994

[Bristow93] Bristow D.J. (Lt.Col. CF), ***Space Command and Control Architectural Infrastructure (SCAI) Air Force/STARS Demonstration Project Management Plan. Appendix B RICC***, Version 2.0, 15 Oct 93

[DemExp95] AF, Loral, ***AF/STARS Demonstration Project Experience Report, Version 2.0 (Draft),*** CDRL A011-002D, December 1994 (currently under Government review)

[IBM93a] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 1 Cleanroom Engineering Process Introduction and Overview***, 31 Jul 93, STARS-05507-001A

[IBM93b] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 2 Organization and Project Formation in the Cleanroom Environment***, 31 Jul 93, STARS-05507-001B

[IBM93c] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 3 Project Execution in the Cleanroom Environment***, 31 Jul 93, STARS-05507-001C

[IBM93d] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 4 Specification Team Practices***, 31 Jul 93, STARS-05507-001D

[IBM93e] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 4 Development Team Practices***, 31 Jul 93, STARS-05507-001E

[IBM93f] IBM, ***STARS Program Cleanroom Engineering Handbook Volume 4 Certification Team Practices***, 31 Jul 93, STARS-05507-001F

[Mills86] Mills H.D., Linger R.C., Hevner A.R., ***Principles of Information Systems Analysis and Design***, Academic Press, 1986

[PAL92a] Arnold P.G. ***Cleanroom Engineering Process***, STARS/SEI Process Asset Library, 1 Oct 92

[PAL92b] Pixton J. Maymir-Ducharme F. ***The TRW Ada Process Model***, STARS/SEI Process Asset Library, 16 Oct 92

[Diaz91-1] Prieto-Diaz, Ruben, Reuse Library Process Model, Loral FS CDRL # 0341-002, 07-26-91

{Diaz91-2] Prieto-Diaz Ruben, Arrango Guillermo, ***Domain Analysis and Software Systems Modeling***, IEEE computer Society Press, 1991

[Royce89] Royce W. ***Ada Process Model***, TRW SEDD Guidebook, 5 Oct 89, SEDD SGB-01-10-89-A

[Royce90a] Royce W., ***TRW's Ada Process Model for Incremental Development of Large Software Systems***, TRW Technologies Series, Jan 1990, TRW-TS-90-01

[Royce90b] Royce W., ***TRW's Ada Process Model for Incremental Development of Large Software Systems***, Proceedings of the 12th International Conference on Software Engineering, Nice France, 26-30 March 1990

LORAL
Federal Systems

SWSC Domain Engineering Experience

AF/STARS SCAI

Brian Bulat

Air Force / STARS Demonstration Project

LORAL
Federal Systems

Overview

- **Domain Engineering Context**
  - SWSC Environment
  - RICC Chip Architecture
  - STARS Megaprogramming
  - SCAI Domain Architecture
  - SCAI Architecture Framework
- **Domain Engineering Lessons Learned**
- **Conclusions**

LORAL
Federal Systems

SWSC Mission

- Supports combat operations mission critical software which meet operational requirements for NORAD, USSPACECOM and AFSPACECOM command and control centers.

- Ensures the operational and technical integrity of national attack warning and space control systems.

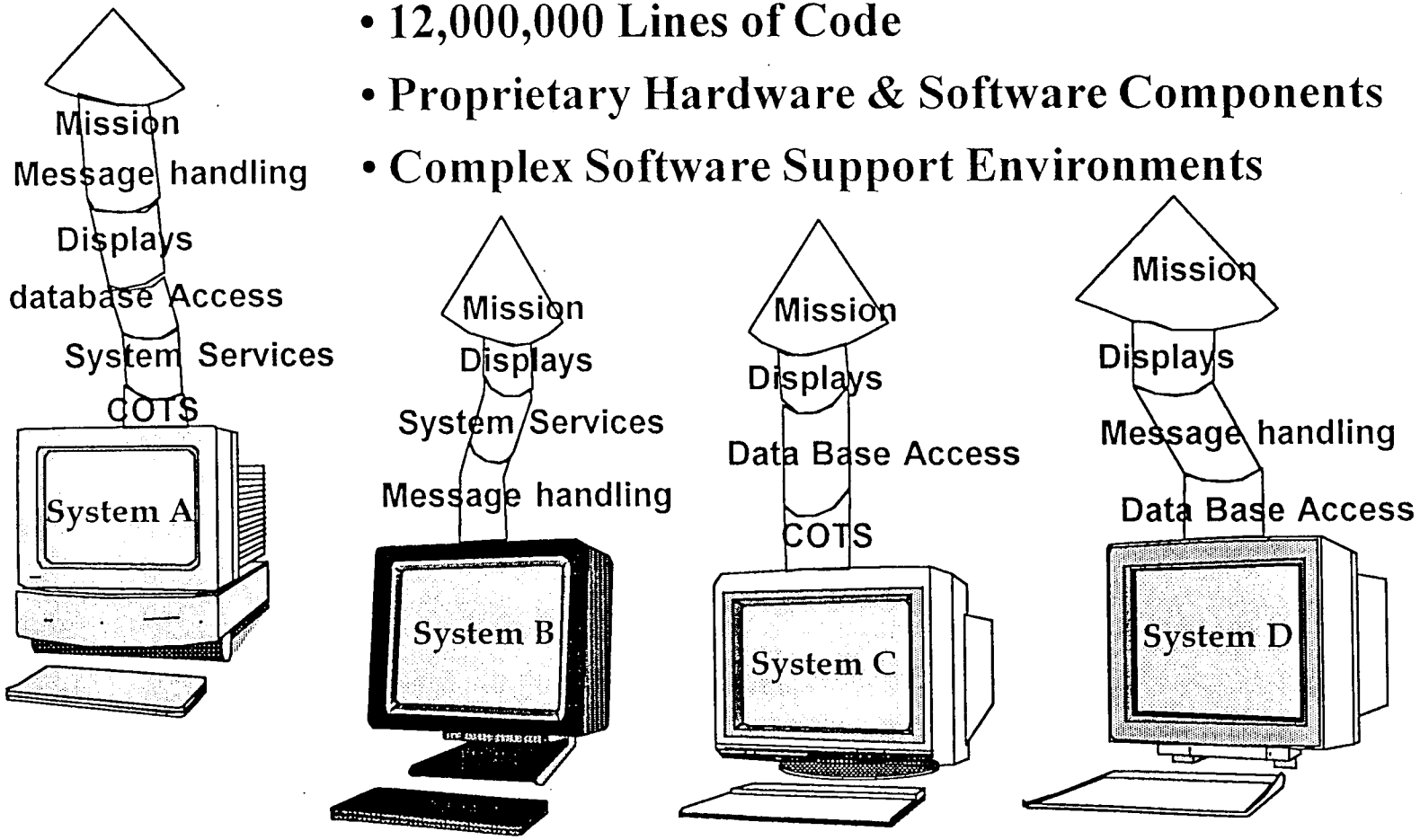
- Provides space and warning software engineering and configuration management.

LORAL
Federal Systems

SWSC Problem

AF/STARS SCAI

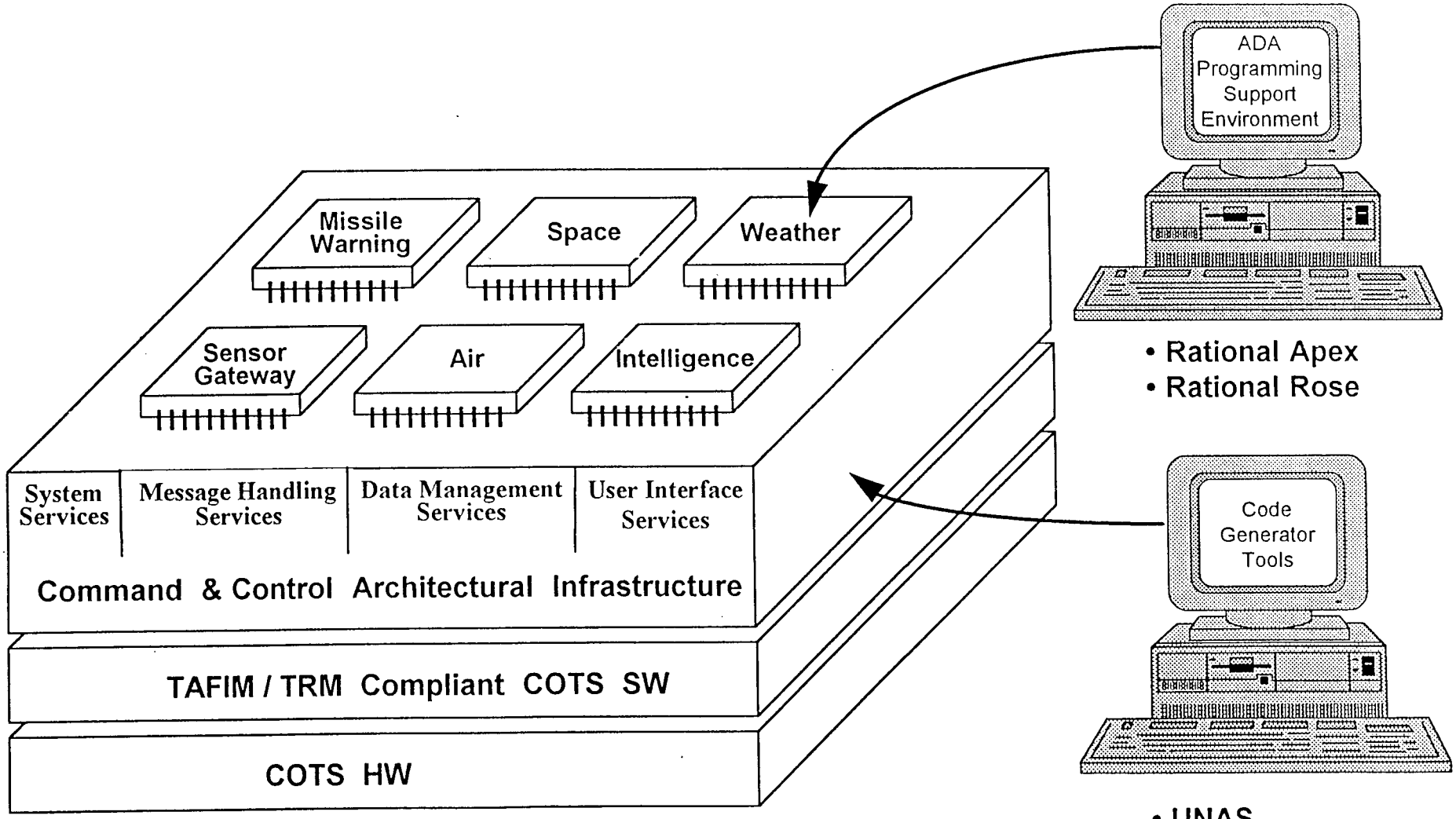
- 34 Separate Operational Systems
- 27 Languages
- 12,000,000 Lines of Code
- Proprietary Hardware & Software Components
- Complex Software Support Environments



LORAL
Federal Systems

Initial C^2 Architectural Goal

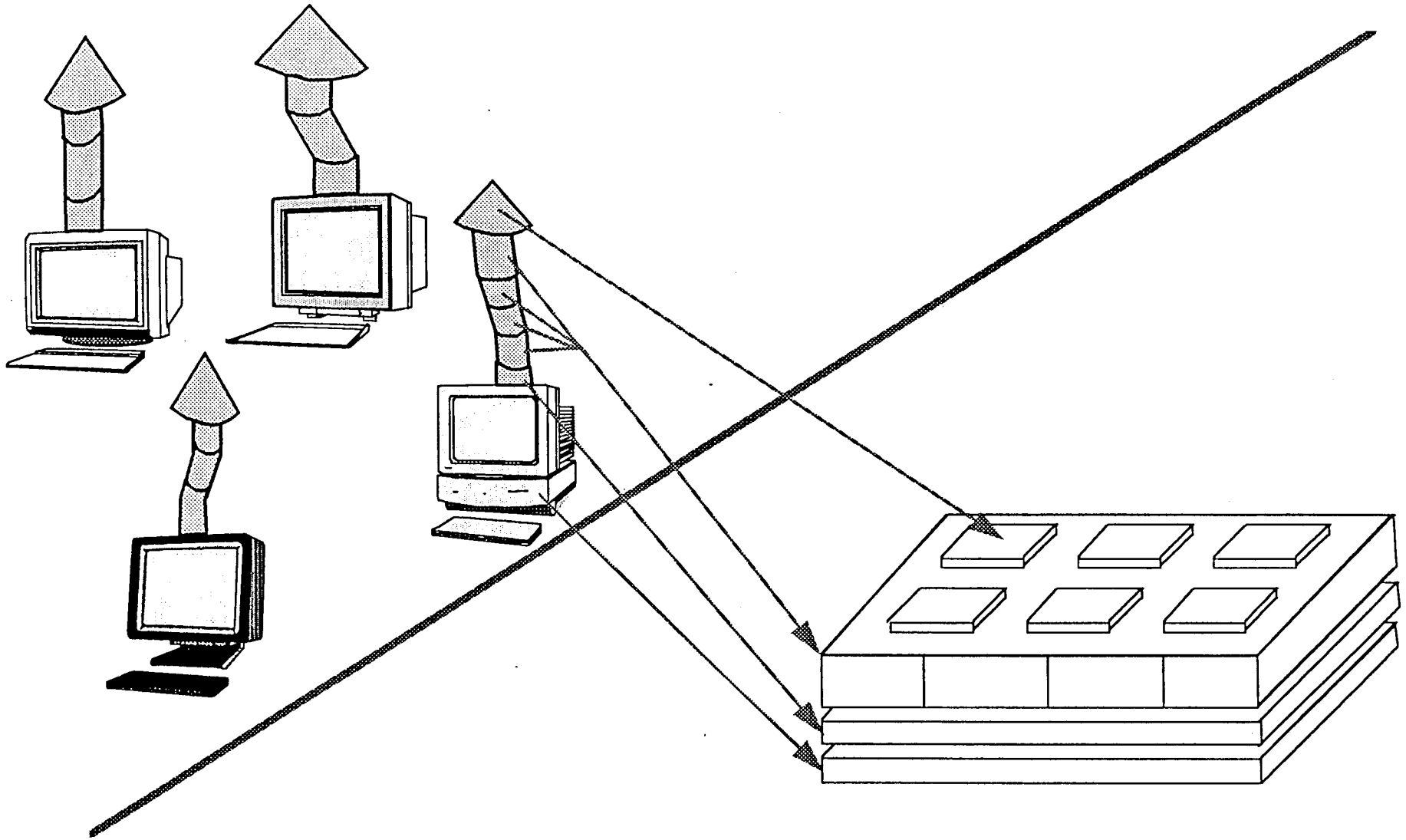
AF/STARS SCAI



LORAL
Federal Systems

Initial Domain Engineering

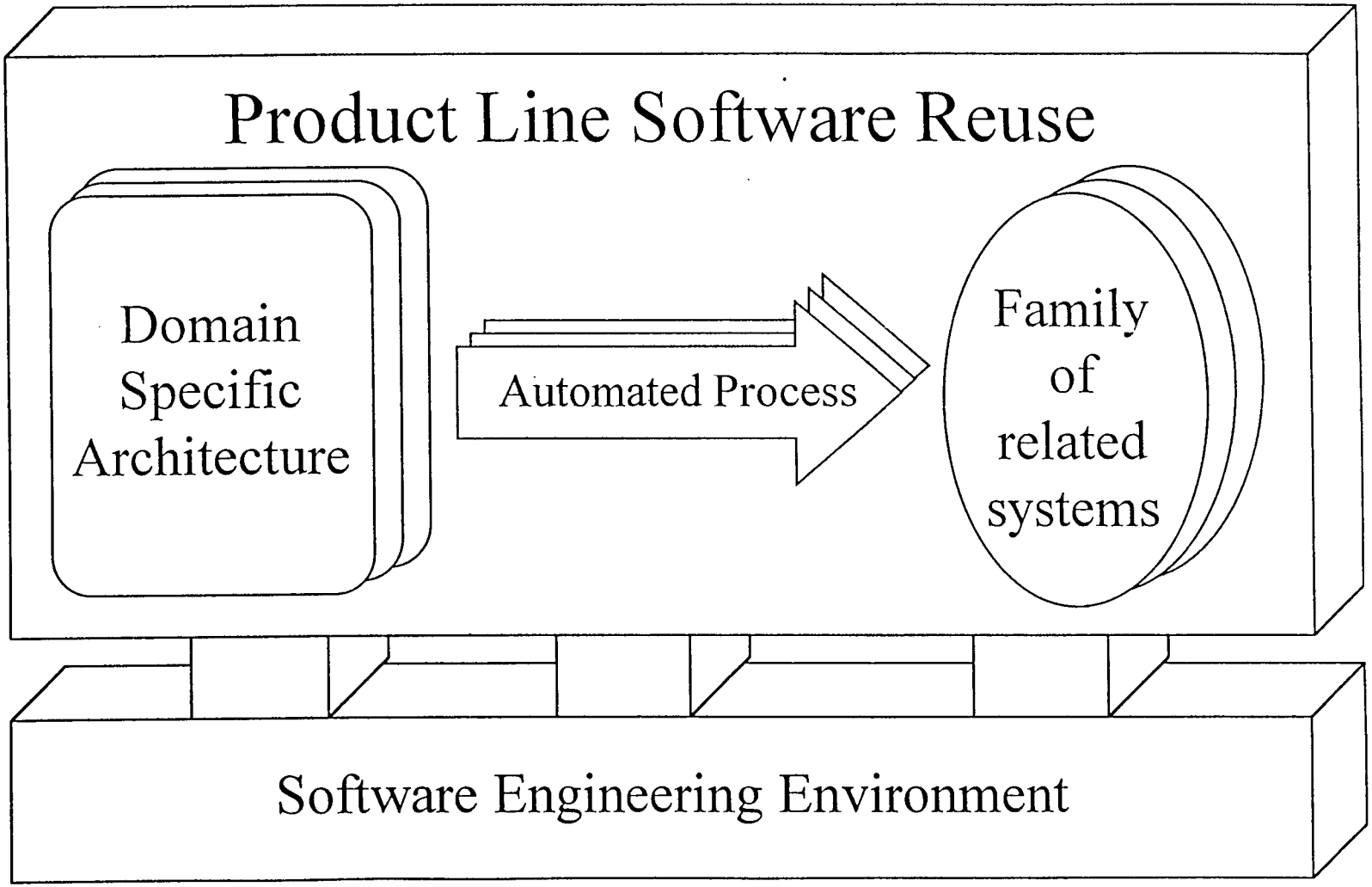
AF/STARS
SCAI



LORAL
Federal Systems

Megaprogramming

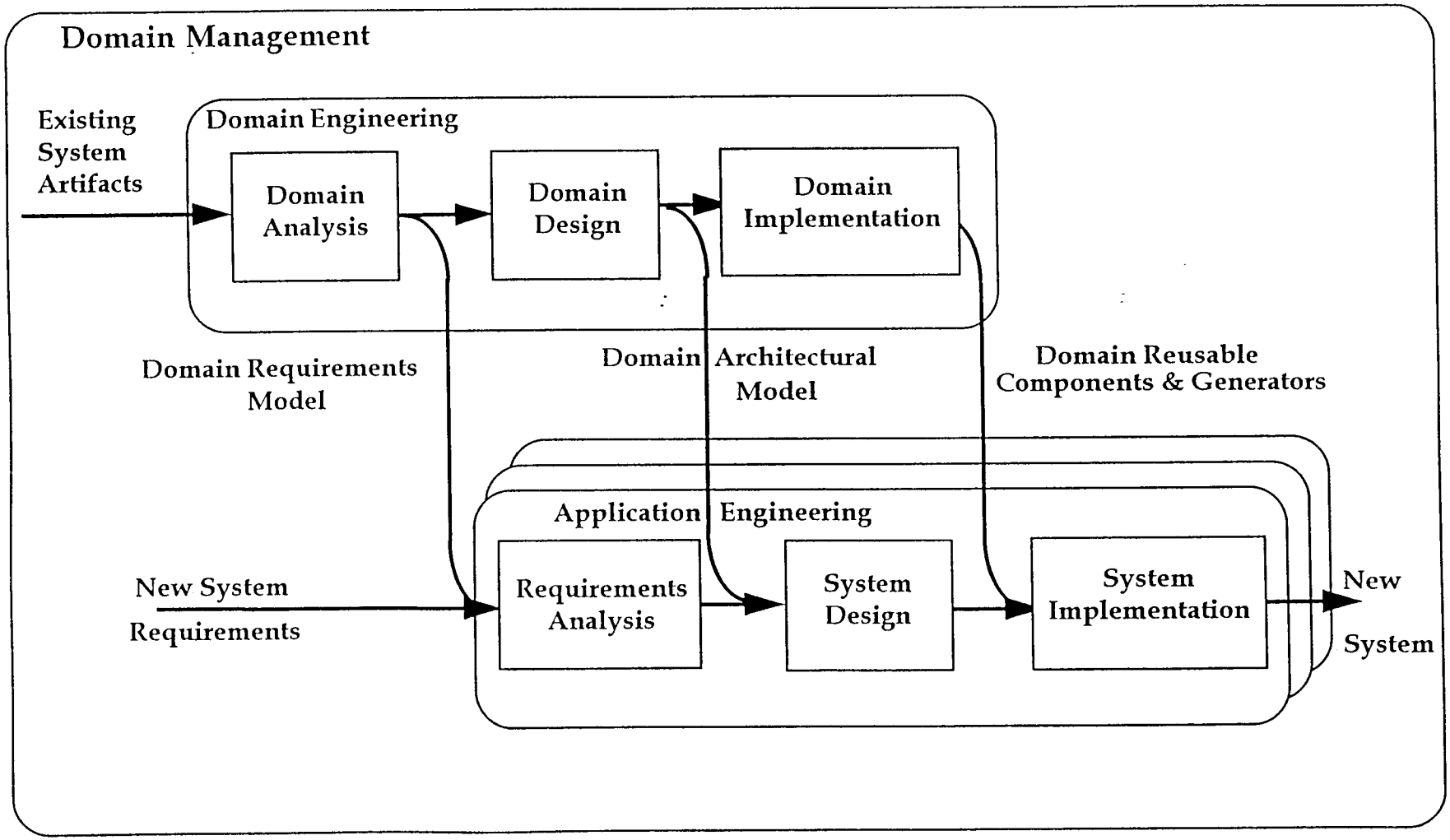
AF/STARS
SCAI



LORAL
Federal Systems

STARS Two Life-Cycle Model

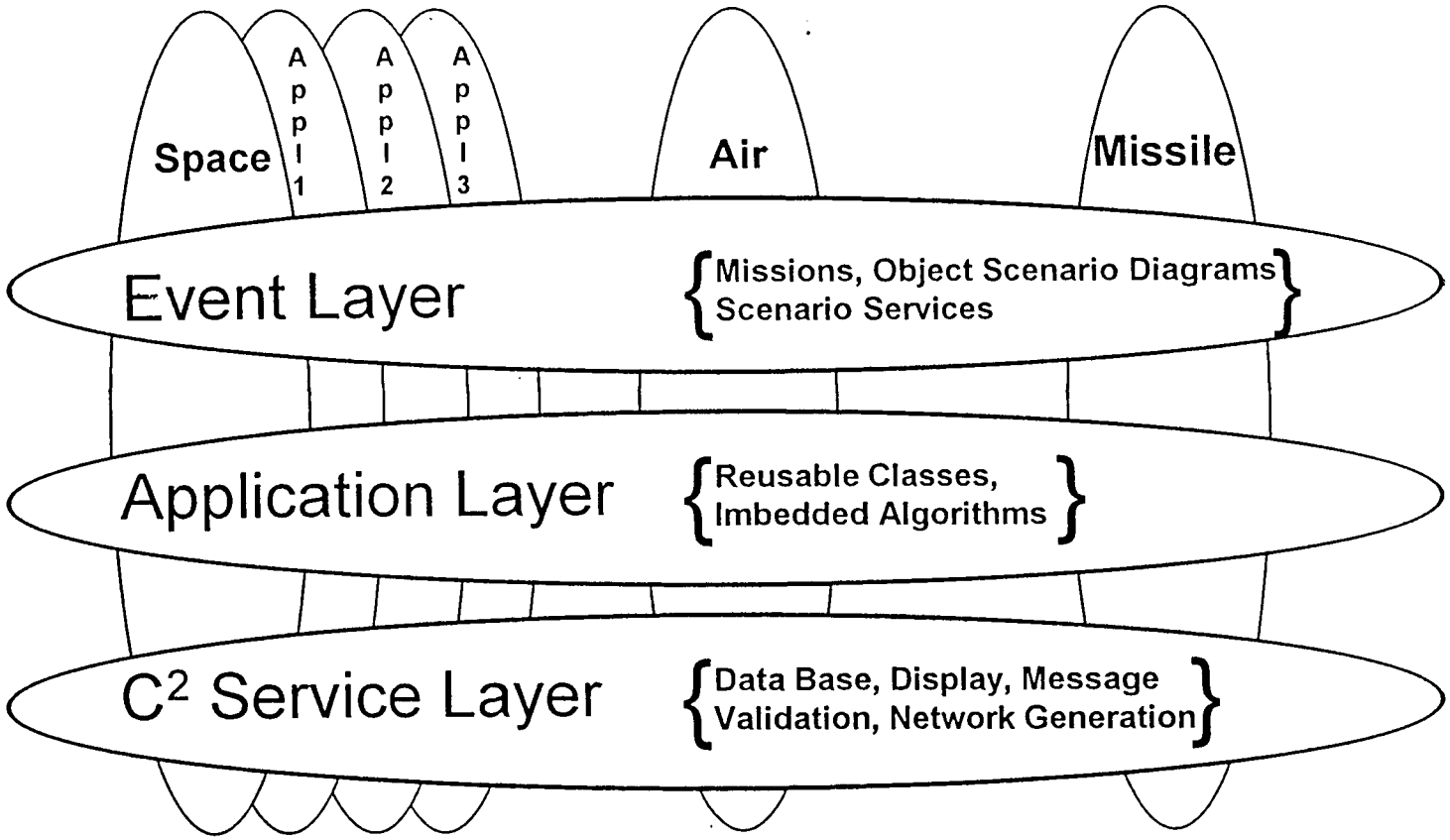
AF/STARS SCAI



LORAL
Federal Systems

Domain Requirements Model

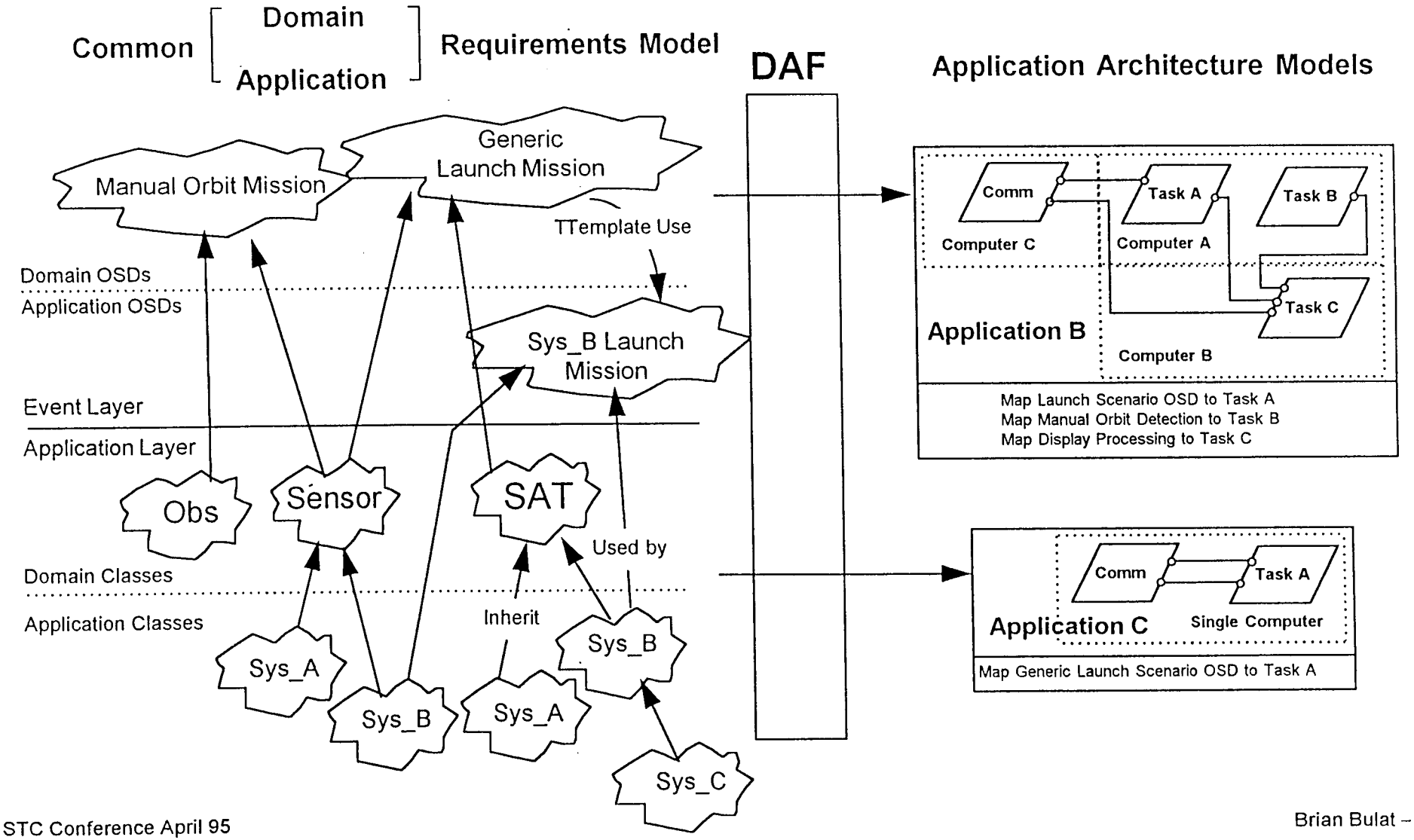
AF/STARS SCAI



LORAL
Federal Systems

Domain Architecture

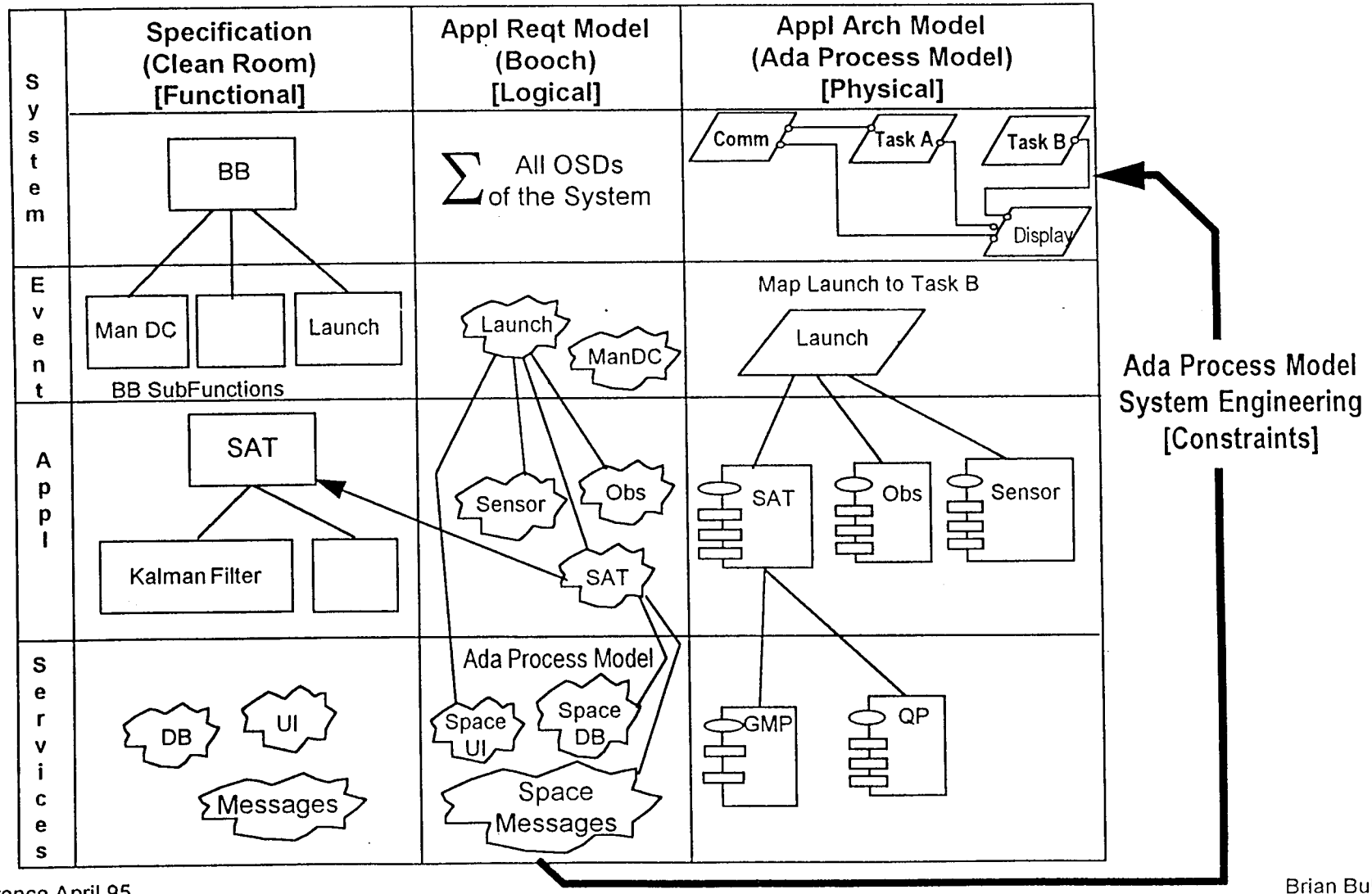
AF/STARS SCAI



LORAL
Federal Systems

Domain Architecture Framework

AF/STARS SCAI

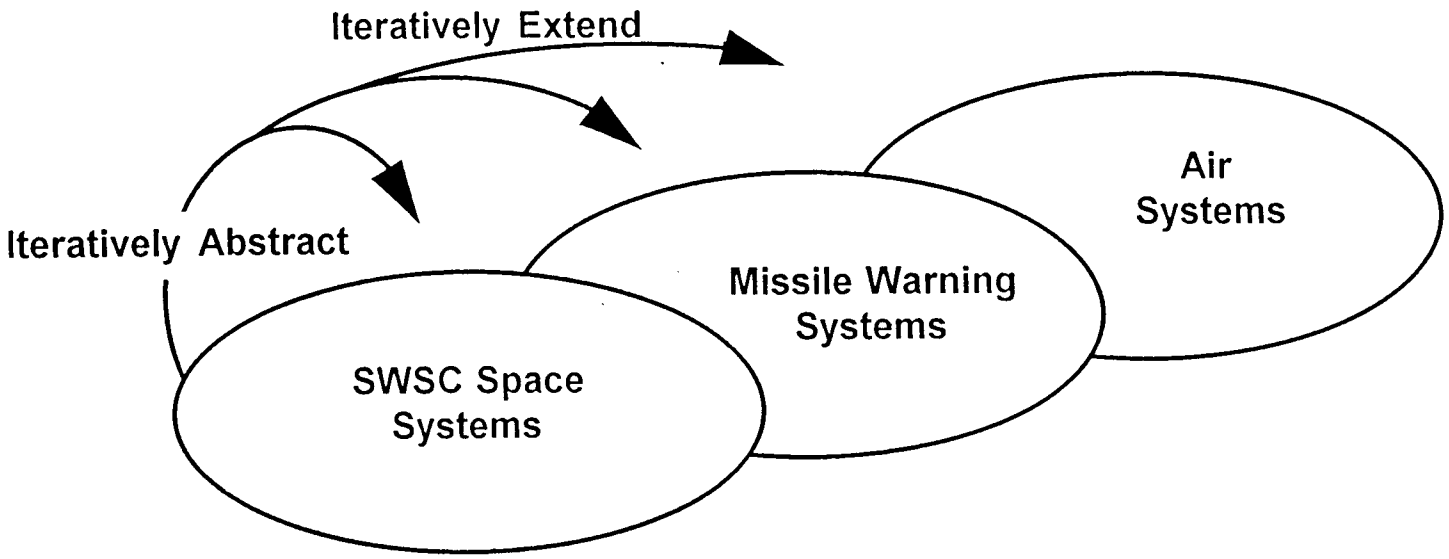


SWSC Domain Engineering Experiences

LORAL Federal Systems

Iterative Domain Engineering

AF/STARS SCAI

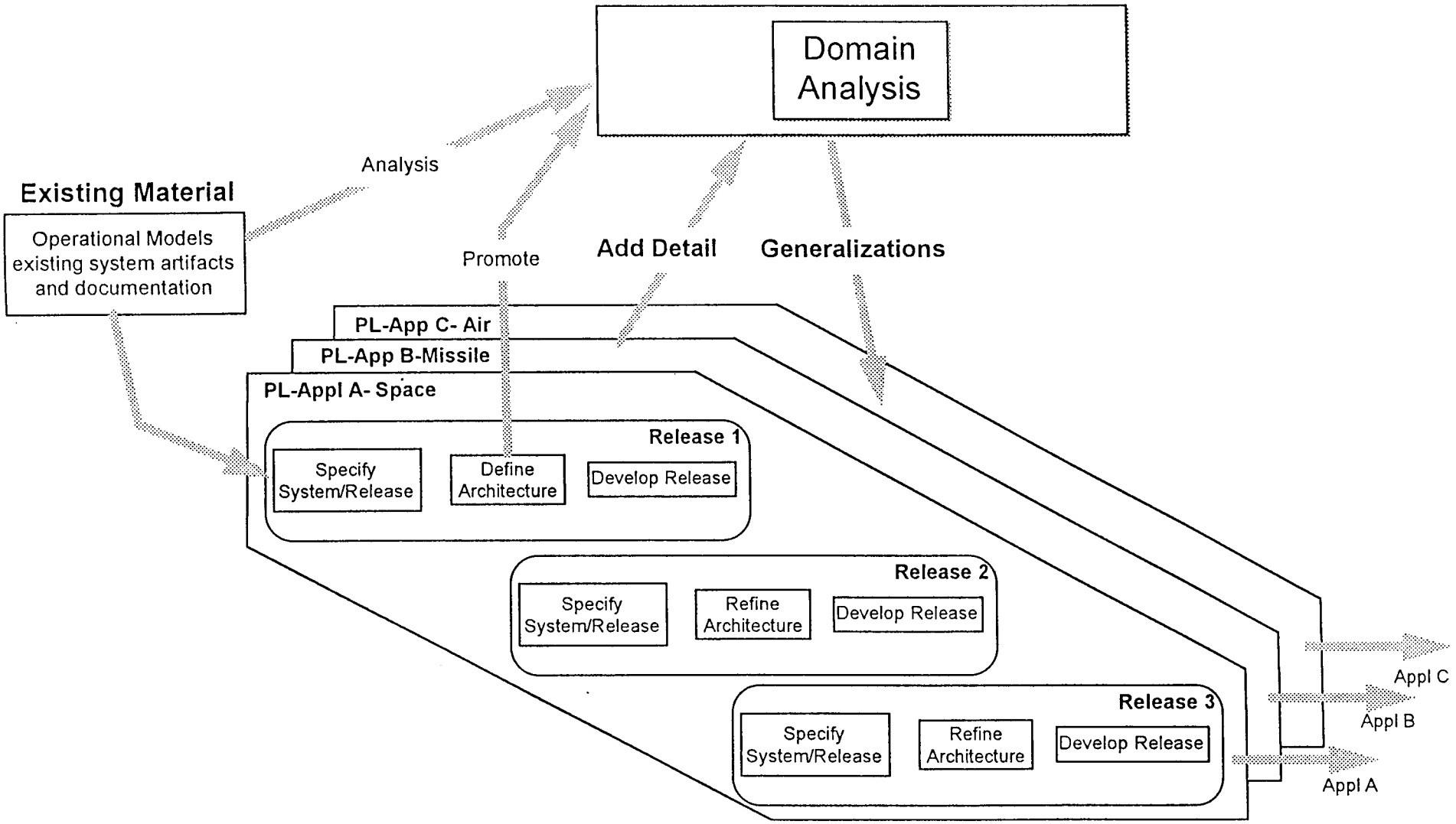


- **Upfront DE Costs**
- **Waterfall life-Cycle Invalid for DE**
- **Very Large C2 Domain**
- **Tightly Couple DE/AE**

LORAL Federal Systems

Modified Two-Lifecycle Model

AF/STARS SCAI



LORAL
Federal Systems

Integrated DE/AE

AF/STARS SCAI

- Common DRM
- Configuration Management Problems
- AE Builds/Tests Components
- Domain Management Costs/Schedules

LORAL
Federal Systems

Product Line Organization Mimics Architecture

- Missions -------- Space Mission Experts
- Applications---- Astronautical Engineers
- Services---------- System Engineers
- AAM---------- System Architects
- Consider People/Jobs when defining architecture

LORAL
Federal Systems

Need Scenarios

- Understand System Dynamics
- Prove Systems can be Constructed from Reusable Components
- Create Simplest Reusable Systems
- Need Validation Data

LORAL
Federal Systems

Analyze Organizations as well as Systems

- SCAI Domain Analysis Restrictions
- Example of Orbit Determination Dialogue
- Keep Dialogues Separate/ User Interface Layers

LORAL
Federal Systems

Need Functional and Object Oriented Mindsets

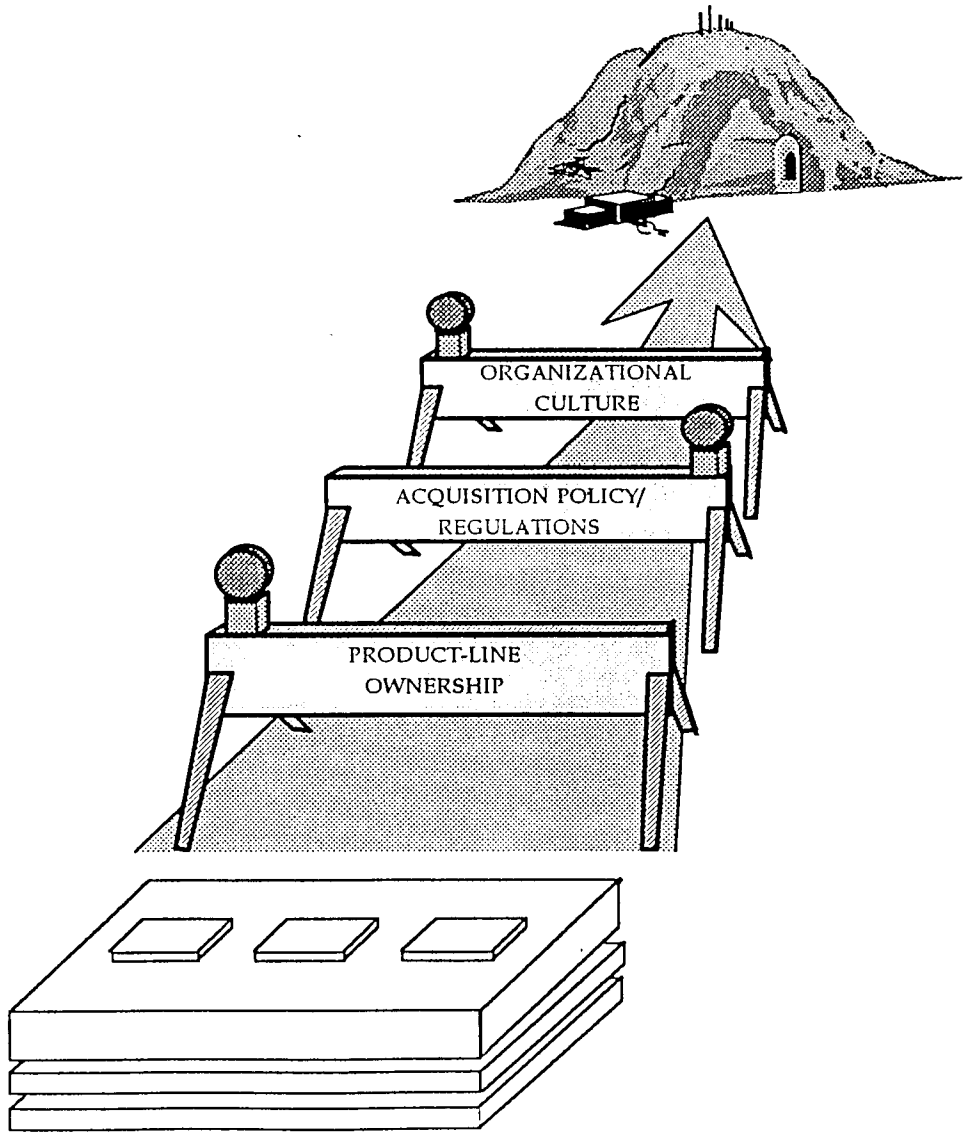
AF/STARS SCAI

- Object Oriented
  - –Domain Understanding
  - –Reuse
- Functional
  - –Formal Methods and Verification
  - –System/Domain Specifications
  - –COTS/GOTS/Technological Change

LORAL
Federal Systems

Need Domain Organization

AF/STARS
SCAI



LORAL
Federal Systems

Summary

AF/STARS SCAI

- SCAI Focus: Use Standard Software Engineering Technologies for DE
- Software Process and Software Architecture are Intimately Related
- Domain Engineering is Iterative and Tightly Coupled with Application Engineering
- Product Line Organizations are a Necessity and Should Mimic the Architecture